Dynamic tags for security protocol

DYNAMIC TAGS FOR SECURITY PROTOCOLS

MYRTO ARAPINIS, STÉPHANIE DELAUNE, AND STEVE KREMER

Abstract. The design and verification of cryptographic protocols is a notoriously difficult task, even in symbolic models which take an abstract view of cryptography. This is mainly due to the fact that protocols may interact with an arbitrary attacker which yields a verification problem that has several sources of unboundedness (size of messages, number of sessions, etc.).

In this paper, we characterize a class of protocols for which deciding security for an unbounded number of sessions is decidable. More precisely, we present a simple transformation which maps a protocol that is secure for a bounded number of protocol sessions (a decidable problem) to a protocol that is secure for an unbounded number of sessions. The precise number of sessions that need to be considered is a function of the security property and we show that for several classical security properties a single session is sufficient. Therefore, in many cases our results yields a design strategy for security protocols: (i) design a protocol intended to be secure for a single session; and (ii) apply our transformation to obtain a protocol which is secure for an unbounded number of sessions.

1. Introduction

Security protocols are distributed programs which aim at guaranteeing properties such as confidentiality of data, authentication of participants, etc. The security of these protocols relies on the one hand on the security of cryptographic primitives, e.g. encryption and digital signatures, and on the other hand on the concurrency-related aspects of the protocols themselves. History has shown that even if cryptography is supposed to be perfect, such as in the classical Dolev-Yao model [20], the correct design of security protocols is notoriously error-prone. See for instance [13] for an early survey on attacks. These difficulties come mainly from two sources of unboundedness: a protocol may be executed several times (we need to consider several protocol *sessions*) and the attacker is allowed to build messages of unbounded size. Indeed, secrecy is known to be undecidable when an unbounded number of sessions is allowed, even if the message size is bounded [21]. However, when the number of sessions is bounded, and even without assuming a bounded message size, the problem becomes co-NP-complete [30]. Moreover, special purpose verification tools (e.g. [4]) exist which are highly efficient when the number of sessions is small.

In this paper we propose a protocol transformation which maps a protocol that is secure for a bounded number of sessions to a protocol that is secure for an unbounded number of sessions. The exact number of sessions that need to be considered depends on the security property under study. We express security properties in a temporal logic with past similar to the logics of [16, 17]. This logic is expressive enough to model security properties such as secrecy and several flavors of non-injective authentication properties. As we will see for

LOGICAL METHODS
IN COMPUTER SCIENCE DOI:10.2168/LMCS-???

these classical security properties verifying a single session will be sufficient and our result provides a strategy to design secure protocols: (i) design a protocol intended to be secure for a single session; and (ii) apply our transformation and obtain a protocol which is secure for an unbounded number of sessions.

Our transformation. Suppose that Π is a protocol between k participants $A_1, \ldots, A_k$. Our transformation adds to Π a preamble in which each participant sends a freshly generated nonce N_i together with his identity to all other participants. This allows each participant to compute a dynamic, session-dependent *tag* $\langle A_1, N_1 \rangle, \ldots, \langle A_k, N_k \rangle$ that will be used to tag each encryption and signature in Π. Our transformation is surprisingly simple and does not require any cryptographic protection of the preamble, i.e., an active attacker is allowed to interfere with this preliminary phase. Intuitively, the security relies on the fact that the participant A_i decides on a given tag for a given session which is ensured to be fresh as it contains his own freshly generated nonce N_i. The transformation is computationally light as it does not add any cryptographic application; it may merely increase the size of messages to be encrypted or signed. The transformation applies to a large class of protocols, which may use symmetric and asymmetric encryption, digital signature and hash functions.

We may note that, *en passant*, we identify a class of tagged protocols for which security is decidable for an unbounded number of sessions. This directly follows from our main result as it stipulates that verifying security for a bounded number of protocol sessions is sufficient to conclude security for an unbounded number of sessions.

Related Work. The kind of compiler we propose here has also been investigated in the area of cryptographic design in computational models, especially for the design of group key exchange protocols. For example, Katz and Yung [23] proposed a compiler which transforms a key exchange protocol secure against a passive eavesdropper into an authenticated protocol which is secure against an active attacker. Earlier work includes compilers for 2-party protocols (e.g. [7]). In the symbolic model, recent works [18, 6] allow one to transform a protocol which is secure in a weak sense (roughly no attacker [18] or just a passive one [6] and a single session) into a protocol secure in the presence of an active attacker and for an unbounded number of sessions. All of these prior works share however a common drawback: the proposed transformations make heavy use of cryptography. This is mainly due to the fact that the security assumptions made on the input protocol are rather weak. As already mentioned in [18], it is important, from an efficiency perspective to lighten the use of cryptographic primitives. In this work, we succeed in doing so at the price of requiring stronger security guarantees on the input protocol. However, we argue that this is acceptable since efficient automatic tools exist to decide this security criterion on the input protocols. Recently, our transformation has also been adapted to the case of offline guessing attacks in password-based protocols [11]. On the one hand the result presented in [11] is more complicated as it considers a more complex security property but, on the other hand, the proof is simplified by the fact that the password is the only secret shared between different sessions.

We can also compare our work with existing decidable protocol classes for an unbounded number of sessions. An early result is the PTIME complexity result by Dolev *et al.* [19] for a restricted class, called *ping-pong* protocols. Other classes have been proposed by Ramanujam and Suresh [28, 29], and Lowe [26]. However, in both cases, temporary secrets,

composed keys and ciphertext forwarding are not allowed which discards protocols (even their tagged version), such as the Yahalom protocol [13].

Different kinds of tags have also been considered in [12, 3, 17, 9, 28]. However these tags are *static* and have a different aim. While our dynamic tagging scheme avoids confusing messages from different sessions, these static tags avoid confusing different messages inside the same session and do not prevent that the same message is reused in two different sessions. Under some additional assumptions (e.g. no temporary secret, no ciphertext forwarding), several decidability results [29, 26] have been obtained by showing that it is sufficient to consider one session per role. But those results cannot deal with protocols which rely on ciphertext forwarding and/or temporary secrets. In the framework we consider here, the question whether such static tags would be sufficient to obtain decidability is still an open question (see [3]). In a similar way, static tags have also been used by Heather et al. [22] to avoid type confusion attacks.

Finally, we may note that our tags are reminiscent of session tags in the UC framework [10] and in particular the method proposed by Barak et al. [5] for computing them. However, in addition to the important differences in the models, these works do not propose a general, systematic transformation which guarantees (joint state) composition between sessions.

This paper can be seen as an extended and enriched version of [2]. In [2], our reduction result was only established for the secrecy property whereas we consider here a larger class of security properties that includes several levels of authentication. Moreover, the proof of our main result is now self-contained and does not rely anymore on the constraint solving procedure presented in [15].

Outline of the paper. Our paper is organized in two parts: Part I presents our result and all the necessary background for the result to be formally stated and Part II is devoted to giving an overview of the proof of the result (for readability some of the more technical proofs are only given in an appendix).

In Part I we first introduce our abstract representation of protocol messages (Section 2) and our formal models for security protocols (Section 3) and properties (Section 4). Next, in Section 5, we formally define our protocol transformation and state our main result which guarantees that attacks only require a bounded number of sessions.

In Part II we give an overview of our proof. In Section 6 we define a transformation on protocol executions and show that a transformed execution *(i)* has several good properties (it is both valid and well-formed), and *(ii)* preserves the satisfaction of attack formulas. In Section 7 we show that we can restrict the sessions that are involved in a valid, well-formed execution while preserving *(i)* validity and well-formedness, and *(ii)* satisfaction of attack formulas. Finally, in Section 8, we use the results from the previous two sections to prove our main result.

— PART I: Presentation of our reduction result —

2. Messages and intruder capabilities

2.1. **Messages.** We use an abstract term algebra to model the messages of a protocol. For this we fix several disjoint sets. We consider an infinite set of *agents* $\mathcal{A} = \{\epsilon, a, b \ldots\}$ with the special agent ϵ standing for the attacker and an infinite set of *agent variables* $\mathcal{X} = \{x_A, x_B, \ldots\}$. We also need to consider an infinite set of *names* $\mathcal{N} = \{n, m \ldots\}$ and an infinite set of *variables* $\mathcal{Y} = \{y, z, \ldots\}$. Among this set of names, we consider the infinite set of names $\mathcal{N}_\epsilon = \{n^\epsilon, \ldots\}$ that corresponds to names known initially by the attacker. We consider the following *signature* $\mathcal{F} = \{\mathsf{encs}/2, \mathsf{enca}/2, \mathsf{sign}/2, \langle\,\rangle/2, \mathsf{h}/1, \mathsf{pub}/1, \mathsf{priv}/1, \mathsf{shk}/2\}$. These function symbols model cryptographic primitives. The symbol $\langle\,\rangle$ represents pairing. The term $\mathsf{encs}(m, k)$ (resp. $\mathsf{enca}(m, k)$) represents the message m encrypted with the symmetric (resp. asymmetric) key k whereas the term $\mathsf{sign}(m, k)$ represents the message m signed by the key k. The function h models a hash function whereas $\mathsf{pub}(a)$ and $\mathsf{priv}(a)$ are used to model the public and the private key respectively of an agent a, and $\mathsf{shk}(a, b)$ ($= \mathsf{shk}(b, a)$) is used to model the long-term symmetric key shared by agents a and b. Names are used to model atomic data such as nonces. The set of *terms* is defined inductively by the following grammar:

$t, t_1, t_2, \ldots$	::=		term
	|	x	agent variable $x \in \mathcal{X}$
	|	a	agent $a \in \mathcal{A}$
	|	y	variable $y \in \mathcal{Y}$
	|	n	name $n \in \mathcal{N}$
	|	$\mathsf{pub}(u)$	application of the symbol pub on $u \in \mathcal{A} \cup \mathcal{X}$
	|	$\mathsf{priv}(u)$	application of the symbol priv on $u \in \mathcal{A} \cup \mathcal{X}$
	|	$\mathsf{shk}(u_1, u_2)$	application of the symbol shk on $u_1, u_2 \in \mathcal{A} \cup \mathcal{X}$
	|	$\mathsf{h}(t)$	application of h
	|	$\mathsf{f}(t_1, t_2)$	application of symbol $\mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \langle\,\rangle\}$

We sometimes write $\langle t_1, \ldots, t_n\rangle$ instead of writing $\langle t_1, \langle \ldots, \langle t_{n-1}, t_n\rangle \ldots\rangle\rangle$. We say that a term is *ground* if it has no variable. We consider the usual notations for manipulating terms. A position p in a term t is a sequence of integers. The empty sequence ε denotes the top-most position. The subterm of t at position p is written $t|_p$. We write $vars(t)$ (resp. $names(t)$, $agents(t)$) for the set of variables (resp. names, agents) occurring in t. We write $\mathsf{St}(t)$ for the set of *syntactic subterms* of a term t and define the set of *cryptographic subterms* of a term t as $\mathsf{CryptSt}(t) = \{\mathsf{f}(t_1, \ldots, t_n) \in \mathsf{St}(t) \mid \mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}\}$. Moreover, we define the set of *long-term keys* as $lgKeys = \{\mathsf{priv}(a) \mid a \in \mathcal{A}\} \cup \{\mathsf{shk}(a, b) \mid a, b \in \mathcal{A}\}$ and the set of *long-term keys of a term* t as

$$lgKeys(t) = \{\mathsf{priv}(u) \mid \mathsf{pub}(u) \in \mathsf{St}(t) \text{ or } \mathsf{priv}(u) \in \mathsf{St}(t)\} \cup \{\mathsf{shk}(u_1, u_2) \in \mathsf{St}(t)\}.$$

and we define $\mathcal{K}_\epsilon = \{\mathsf{priv}(\epsilon)\} \cup \{\mathsf{shk}(a, \epsilon) \mid a \in \mathcal{A}\}$. Intuitively $\mathcal{K}_\epsilon$ represents the set of long-term keys of the attacker. An *atom* is a long-term key, a name or a variable.

We define the set of *plaintexts* of a term t as the set of atoms that occur in plaintext position, i.e.

- $plaintext(\mathsf{h}(u)) = plaintext(\mathsf{f}(u, v)) = plaintext(u)$ for $\mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}\}$,

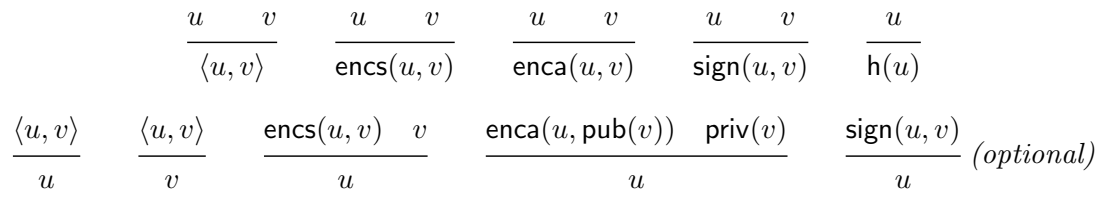

Figure 1: Intruder deduction system.

- $plaintext(\langle u, v\rangle) = plaintext(u) \cup plaintext(v)$, and
- $plaintext(u) = \{u\}$ otherwise.

All these notions are extended to sets of terms and to other kinds of term containers as expected. We denote by $\#S$ the cardinality of a set S. Substitutions are written $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where its *domain* is $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$. The substitution σ is *ground* if all the t_i are ground. The application of a substitution σ to a term t is written $\sigma(t)$ or $t\sigma$. Two terms t_1 and t_2 are *unifiable* if $t_1\sigma = t_2\sigma$ for some substitution σ, that is called a *unifier*. We denote by $\mathrm{mgu}(t_1, t_2)$ the *most general unifier* of t_1 and t_2.

Example 1. Let $t = \mathsf{encs}(\langle n, a\rangle, \mathsf{shk}(a, b))$. We have that $vars(t) = \emptyset$, i.e. t is ground, $names(t) = \{n\}$, $agents(t) = \{a, b\}$, $lgKeys(t) = \{\mathsf{shk}(a, b)\}$, $plaintext(t) = \{n, a\}$, and $\mathsf{St}(t) = \{t, \langle n, a\rangle, \mathsf{shk}(a, b), n, a\}$. The terms $\mathsf{shk}(a, b)$, a, n and $\mathsf{priv}(a)$ are atoms.

2.2. **Intruder capabilities.** We model the intruder's abilities to construct new messages by the deduction system given in Figure 1. The first line describes the *composition rules*. The second line describes the *decomposition rules*. The intuitive meaning of these rules is that an intruder can compose new messages by pairing, encrypting, signing and hashing previously known messages provided he has the corresponding keys. Conversely, he can decompose messages by projecting or decrypting provided he has the decryption keys. Our optional rule expresses that an intruder can retrieve the whole message from its signature. Whether this property holds depends on the actual signature scheme. Therefore we consider this rule to be optional. Our results hold in both cases.

Definition 1 (deducible)**.** We say that a term u is *deducible* from a set of terms T, denoted $T \vdash u$, if there exists a tree such that its root is labeled by u, its leaves are labeled with $v \in T \cup \mathcal{A} \cup \mathcal{N}_\epsilon \cup \mathcal{K}_\epsilon \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\}$ and for every node labeled by v having n sons labeled by $v_1, \ldots, v_n$ we have that $\dfrac{v_1 \ \ldots \ v_n}{v}$ is an instance of one of the inference rules given in Figure 1.

Example 2. The term $\langle n, \mathsf{shk}(a, b)\rangle$ is deducible from $\{\mathsf{encs}(n, \mathsf{shk}(a, b)), \mathsf{shk}(a, b)\}$.

We are now able to state the following lemma that can be easily proved by induction on the proof tree witnessing $T \vdash t$.

Lemma 1. Let T be a set of terms and t be a term such that $T \vdash t$. We have that:

$$plaintext(t) \ \subseteq \ plaintext(T) \cup \mathcal{A} \cup \mathcal{N}_\epsilon \cup \mathcal{K}_\epsilon \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\}.$$

3. Model for security protocols

In this section, we give a language for specifying protocols and define their execution in the presence of an active attacker. Our model is similar to existing ones (see e.g. [30, 17]).

3.1. **Syntax.** We consider protocols specified in a language allowing parties to exchange messages built from identities and randomly generated nonces using pairing, public key, symmetric encryption, hashing and digital signatures. The individual behavior of each protocol participant is defined by a *role* describing a sequence of *events*. The main events we consider are *communication events* (i.e. message receptions and message transmissions) and *status events* to mark different stages reached by the protocol. These status events will help us specify a large class of security properties (a logic of properties is given in Section 4). These are issued by participants to denote their current state in the execution of a protocol role.

Definition 2 (event)**.** An *event* is either

- a *communication event*, i.e. a message reception, denoted by $\mathsf{rcv}(m)$ or a message transmission, denoted by $\mathsf{snd}(m)$, where m is a term; or
- a *status event* of the form $\mathsf{P}(t_1, \ldots, t_n)$ where each t_i is a term (not necessarily ground) and $\mathsf{P} \in \mathcal{P}$ is a predicate symbol of arity n.

Typically, status events give information about the state of the principal. For instance, we will consider a status event that indicates that the principal has started or finished a session. The set of variables of an event is defined as expected, considering all the terms occurring in the event's specification.

Definition 3 (roles)**.** A role is of the form $\lambda x_1 \ldots \lambda x_k . \nu y_1 \ldots \nu y_p .\ \mathsf{seq}$, where:

- $X = \{x_1, \ldots, x_k\}$ is a set of agent variables, i.e. the parameters of the role corresponding to the k participants of the protocol,
- $Y = \{y_1, \ldots, y_p\}$ is a set of variables: the nonces generated by the role,
- $\mathsf{seq} = \mathsf{e}_1; \mathsf{e}_2; \ldots; \mathsf{e}_\ell$ is a sequence of events such that $(\mathit{vars}(\mathsf{seq}) \setminus X) \subseteq \mathcal{Y}$, i.e. all agent variables are parameters.

Moreover, we have that:

(1) seq satisfies the *origination property*, that is for any send or status event e_i, for any variable $x \in \mathit{vars}(\mathsf{e}_i) \setminus (X \cup Y)$, we have that $x \in \mathit{vars}(\mathsf{e}_j)$ for some receive event e_j where $j < i$; and

(2) seq satisfies the *plaintext origination property*, that is for any send or status event e_i, for any variable $x \in \mathit{plaintext}(\mathsf{e}_i) \setminus (X \cup Y)$, we have that $x \in \mathit{plaintext}(\mathsf{e}_j)$ for some receive event e_j where $j < i$.

The set of roles is denoted by Roles. The *length* of a role is the number of elements in its sequence of events. A *k-party protocol* is a mapping $\Pi : [k] \to \mathsf{Roles}$, where $[k] = \{1, 2, \ldots, k\}$.

The condition (1) above ensures that each variable which appears in a send or status event is a nonce, a parameter, or a variable that has been introduced in a previously received message. Condition (2) ensures that a key used for encrypting or signing cannot be extracted and used as plaintext, e.g. forbidding a sequence $\mathsf{rcv}(\mathsf{encs}(y, z)); \mathsf{snd}(z)$.

Example 3. We illustrate our protocol syntax on the familiar Needham-Schroeder public-key protocol [27]. In our syntax this protocol is modeled as follows.

$$\begin{array}{lll} \Pi(1) = & \lambda x_A.\lambda x_B.\nu y. & \\ & \mathsf{snd}(\mathsf{enca}(\langle y, x_A\rangle, \mathsf{pub}(x_B))); & \\ & \mathsf{rcv}(\mathsf{enca}(\langle y, z\rangle, \mathsf{pub}(x_A))); & \\ & \mathsf{snd}(\mathsf{enca}(z, \mathsf{pub}(x_B))) & \end{array} \qquad \begin{array}{lll} \Pi(2) = & \lambda x_A.\lambda x_B.\nu y'. & \\ & \mathsf{rcv}(\mathsf{enca}(\langle z', x_A\rangle, \mathsf{pub}(x_B))); & \\ & \mathsf{snd}(\mathsf{enca}(\langle z', y'\rangle, \mathsf{pub}(x_A))); & \\ & \mathsf{rcv}(\mathsf{enca}(y', \mathsf{pub}(x_B))) & \end{array}$$

The initiator, role $\Pi(1)$ played by x_A, sends to the responder, role $\Pi(2)$ played by x_B, his identity together with a freshly generated nonce y, encrypted with the responder's public key. The responder replies by copying the initiator's nonce and adds a fresh nonce y', encrypted by the initiator's public key. The initiator acknowledges by forwarding the responder's nonce encrypted by his public key.

Clearly, not all protocols written using the syntax above are meaningful. In particular, some of them might not be *executable*. For instance, a k-party protocol where $\Pi(1) := \mathsf{rcv}(\mathsf{h}(x)); \mathsf{snd}(x)$ is not executable since an agent is not able to extract the content of a hash. A precise definition of executability is not relevant for our result. We only need to consider the weaker plaintext origination hypothesis (Condition 2 stated in Definition 3). In particular, our result also holds for non-executable protocols such as the one given above.

3.2. **Semantics.** In our model, a session corresponds to the instantiation of one role. This means in particular that one "normal execution" of a k-party protocol requires k sessions, one per role[1]. We may want to consider several sessions corresponding to different instantiations of a same role. Since the adversary may block, redirect and send new messages, all the sessions might be interleaved in many ways. Such an interleaving is captured by the notion of a *scenario*.

Definition 4 (scenario)**.** A *scenario for a protocol* $\Pi : [k] \to \mathsf{Roles}$ is a sequence $\mathsf{sc} = (r_1, sid_1)\cdots(r_n, sid_n)$ where r_i is a role and sid_i a session identifier such that $1 \leq r_i \leq k$, $sid_i \in \mathbb{N} \setminus \{0\}$, the number of identical occurrences of a pair (r, sid) is smaller than the length of the role r, and $sid_i = sid_j$ implies $r_i = r_j$.

The condition on identical occurrences ensures that a role cannot execute more events than it contains. The last condition ensures that a session number is not reused by other roles. We say that $(r, s) \in \mathsf{sc}$ if (r, s) is an element of the sequence sc.

Given a scenario and an instantiation for the parameters, we define a *symbolic trace*, that is a sequence of events that corresponds to the interleaving of the scenario, for which the parameters have been instantiated, fresh nonces are generated and variables are renamed to avoid name collisions between different sessions.

Definition 5 (symbolic trace)**.** Let Π be a k-party protocol with

$$\Pi(j) = \lambda x_1^j.\ldots\lambda x_k^j.\nu y_1^j.\ldots\nu y_{p_j}^j.\mathsf{e}_1^j;\ldots;\mathsf{e}_{\ell_j}^j \qquad \text{for } 1 \leq j \leq k.$$

Given a scenario $\mathsf{sc} = (r_1, sid_1)\cdots(r_n, sid_n)$ and a function $\alpha : \mathbb{N} \to \mathcal{A}^k$, the *symbolic trace* $\mathsf{tr} = \mathsf{e}_1^{sid_1};\ldots;\mathsf{e}_n^{sid_n}$ *associated to* sc *and* α is defined as follows.

Let $q_i = \#\{j \mid j \leq i, (r_j, sid_j) \in \mathsf{sc}, \text{ and } sid_j = sid_i\}$, i.e. the number of occurrences up to this point in sc of the session sid_i. We have that $q_i \leq \ell_{r_i}$ and $\mathsf{e}_i = (\mathsf{e}_{q_i}^{r_i})\sigma_{r_i,sid_i}$, where $\mathrm{dom}(\sigma_{r,sid}) = vars(\Pi(r))$ and

[1]In the literature, the word session is often used in an abusive way to represent an execution of the *protocol*, i.e. one session per role, whereas we use it for the execution of a *role*.

- $\sigma_{r,sid}(y) = n_y^{sid}$ if $y \in \{y_1^r, \ldots, y_{p_r}^r\}$, where n_y^{sid} is a fresh name from $\mathcal{N}$;
- $\sigma_{r,sid}(x_i^r) = a_i$ when $\alpha(sid) = (a_1, \ldots, a_k)$;
- $\sigma_{r,sid}(z) = z^{sid}$ otherwise, where z^{sid} is a fresh variable.

A session *sid* is said to be *dishonest* w.r.t. α and a set of ground atoms T_0 when $\alpha(sid) = (a_1, \ldots, a_k)$ and $T_0 \vdash \mathsf{priv}(a_i)$ or $T_0 \vdash \mathsf{shk}(a_i, v)$ for some $v \neq \epsilon$ and $1 \leq i \leq k$.

Intuitively, a session *sid* is honest if all of its participants, from the point of view of the agent playing the session *sid*, are honest (i.e. they are neither the attacker ϵ nor did they disclose their long-term keys). Note that since all agent variables occurring in a role, occur as parameters of this role (see Definition 3), a symbolic trace does not contain agent variables.

The notational conventions we use for names and variables occurring in a symbolic trace (*e.g.* n_y^{sid} and z^{sid}) are not really relevant to state our main result. However, we will rely on this notation in Part II when we prove our reduction result.

Example 4. Consider again the Needham-Schroeder protocol. Let $\Pi(1)$ and $\Pi(2)$ be the two roles introduced in Example 3. Let s_1 and s_2 be two sessions numbers ($s_1 \neq s_2$), $\mathsf{sc} = (1, s_1)(2, s_2)(2, s_2)(1, s_1)(1, s_1)$ and α the function such that $\mathrm{dom}(\alpha) = \{s_1, s_2\}$, $\alpha(s_1) = (a, c)$, and $\alpha(s_2) = (a, b)$. This is the scenario allowing us to retrieve the famous attack due to Lowe [24]. The symbolic trace associated to Π, sc, and α is given below:

$$\begin{array}{rcl} \mathsf{tr} & = & \mathsf{snd}(\mathsf{enca}(\langle n_y^{s_1}, a\rangle, \mathsf{pub}(c)));\\ & & \mathsf{rcv}(\mathsf{enca}(\langle z'^{s_2}, a\rangle, \mathsf{pub}(b)));\ \mathsf{snd}(\mathsf{enca}(\langle z'^{s_2}, n_{y'}^{s_2}\rangle, \mathsf{pub}(a)));\\ & & \mathsf{rcv}(\mathsf{enca}(\langle n_y^{s_1}, z^{s_1}\rangle, \mathsf{pub}(a)));\ \mathsf{snd}(\mathsf{enca}(z^{s_1}, \mathsf{pub}(c))) \end{array}$$

An *execution trace* is an instance of such a symbolic trace. Appending an event e to an execution trace exec is written $\mathsf{exec}; \mathsf{e}$. The function length has the usual meaning: $\mathsf{length}([\,]) = 0$ and $\mathsf{length}(\mathsf{exec}; \mathsf{e}) = 1 + \mathsf{length}(\mathsf{exec})$. The prefix of an execution trace consisting of the first i events is denoted as exec_i, with $\mathsf{exec}_0 = [\,]$ and $\mathsf{exec}_n = \mathsf{exec}$ when $n \geq \mathsf{length}(\mathsf{exec})$.

Definition 6 (knowledge of an execution trace exec)**.** Let exec be an execution trace. The knowledge of exec is the set of terms given by $\mathsf{K}(\mathsf{exec}) = \{u \mid \mathsf{snd}(u) \in \mathsf{exec}\}$.

As usual, we are only interested in *valid* execution traces - those traces where the attacker only sends messages that he can compute from his initial knowledge and the messages he has seen on the network.

Definition 7 (valid execution trace)**.** Let T_0 be a set of ground terms (intuitively T_0 represents the initial knowledge of the attacker). A ground execution trace $\mathsf{exec} = \mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}$ is *valid* w.r.t. T_0 if for all $1 \leq i \leq \ell$, whenever $\mathsf{e}_i = \mathsf{rcv}(m)$, we have that $T_0 \cup \mathsf{K}(\mathsf{exec}_i) \vdash m$.

Example 5. Let $T_0 = \{a, b, c, \mathsf{priv}(c)\}$. Let tr be the symbolic trace described in Example 4 and $\sigma = \{z^{s_1} \mapsto n_{y'}^{s_2},\ z'^{s_2} \mapsto n_y^{s_1}\}$. The execution trace $\mathsf{tr}\sigma$ is valid w.r.t. T_0. Indeed, we have that

- $T_1 \stackrel{\mathsf{def}}{=} T_0 \cup \{\mathsf{enca}(\langle n_y^{s_1}, a\rangle, \mathsf{pub}(c))\} \vdash \mathsf{enca}(\langle n_y^{s_1}, a\rangle, \mathsf{pub}(b))$, and
- $T_1 \cup \{\mathsf{enca}(\langle n_y^{s_1}, n_{y'}^{s_2}\rangle, \mathsf{pub}(a))\} \vdash \mathsf{enca}(\langle n_y^{s_1}, n_{y'}^{s_2}\rangle, \mathsf{pub}(a))$.

The purpose of the following lemma is to characterize the terms that occur in plaintext position in a valid execution. Intuitively, the lemma states that any plaintext occurring in a valid execution either occurs as a plaintext in the underlying symbolic trace, or was known by the attacker since the beginning, i.e., is part of the attacker's initial knowledge.

Lemma 2. Let Π be a k-party protocol and $\mathsf{tr} = [\mathsf{ee}_1^{sid_1}; \ldots; \mathsf{ee}_\ell^{sid_\ell}]$ be a symbolic trace associated to it. Let T_0 be a set of ground atoms, and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a valid execution trace associated to tr (w.r.t. T_0). We have that:

$$\mathit{plaintext}(\mathsf{exec}) \subseteq \mathit{plaintext}(\mathsf{tr}) \cup T_0 \cup \mathcal{N}_\epsilon \cup \mathcal{K}_\epsilon \cup \mathcal{A} \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\}.$$

This lemma can be shown by induction on the length of the underlying symbolic trace. We rely on Lemma 1 to deal with the case of a receive event, and on the plaintext origination property (Condition (2) in Definition 3) to deal with the case of a status or a send event.

4. Security properties

In this section, we propose a logic for specifying security properties. Our logic is similar to existing ones (see e.g [16, 17]). In particular, it is expressive enough to specify security properties like secrecy and different forms of authentication including aliveness, weak agreement and non-injective agreement. Its semantics is defined as usual on execution traces.

4.1. A logic for security properties.

As in [17], status events are used to specify security properties while the other events describe the execution of the protocol. We only consider one temporal operator and this operator should only concern status events. That is why we divide the logic into two layers.

Definition 8. A formula of $\mathcal{L}$ is an expression ϕ defined by the following grammar:

$$\begin{aligned} \phi, \phi_i &:= \mathsf{learn}(u_0) \mid \neg\phi \mid \exists x.\phi \mid \phi_1 \vee \phi_2 \mid \mathsf{C}(u) \mid \Diamond\,\psi \mid \psi \\ \psi, \psi_i &:= \mathsf{true} \mid \mathsf{P}(u_1, \ldots, u_n) \mid \neg\psi \mid \psi_1 \vee \psi_2 \end{aligned}$$

where $u_0, u_1, \ldots, u_n$ are terms and $u \in \mathcal{A} \cup \mathcal{X}$.

Standard formulas true, $\neg\phi$, and $\phi_1 \vee \phi_2$ carry the usual meaning. The formula $\mathsf{learn}(u_0)$ states that the attacker knows the term u_0, whereas $\mathsf{P}(u_1, \ldots, u_n)$ is a status event. The formula $\mathsf{C}(u)$ states that the agent u is compromised (his secret keys are known to the attacker). The formula $\Diamond\,\psi$ means that 'ψ held in the past'. When x is a variable, we write $\exists x.\phi$ to bind x in ϕ, with the quantifier carrying the usual meaning. Other operators can be represented using the above defined operators. For instance, the abbreviations $\mathsf{NC}(u)$, false, $\wedge$, $\forall$, and $\Rightarrow$ are defined by $\mathsf{NC}(u) \stackrel{\mathsf{def}}{=} \neg\mathsf{C}(u)$, $\mathsf{false} \stackrel{\mathsf{def}}{=} \neg\mathsf{true}$, $\phi_1 \wedge \phi_2 \stackrel{\mathsf{def}}{=} \neg(\neg\phi_1 \vee \neg\phi_2)$, $\forall x.\phi \stackrel{\mathsf{def}}{=} \neg\exists x.\neg\phi$, and $\phi_1 \Rightarrow \phi_2 \stackrel{\mathsf{def}}{=} \neg\phi_1 \vee \phi_2$.

In the sequel, we assume that formulas are *closed*, i.e. they contain no free variables, and that each variable is quantified at most once (this can be easily ensured by using renaming). We also assume that the variables occurring in a formula ϕ are disjoint from the variables occurring in the considered symbolic trace.

Formulas are interpreted at some position along an execution trace as stated in Definition 9.

Definition 9 (concrete validity)**.** Let ϕ be a closed formula in $\mathcal{L}$, exec be a ground execution trace and T_0 be a set of ground terms. We define $\langle \mathsf{exec}, T_0 \rangle \models \phi$ as:

$\langle \mathsf{exec}, T_0 \rangle \models \mathsf{true}$		
$\langle \mathsf{exec}, T_0 \rangle \models \mathsf{learn}(m)$	iff	$T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash m$
$\langle \mathsf{exec}, T_0 \rangle \models \neg\phi$	iff	$\langle \mathsf{exec}, T_0 \rangle \not\models \phi$
$\langle \mathsf{exec}, T_0 \rangle \models \phi_1 \vee \phi_2$	iff	$\langle \mathsf{exec}, T_0 \rangle \models \phi_1$ or else $\langle \mathsf{exec}, T_0 \rangle \models \phi_2$
$\langle \mathsf{exec}, T_0 \rangle \models \exists x.\phi$	iff	there exists a ground term t s.t. $\langle \mathsf{exec}, T_0 \rangle \models \phi\{x \mapsto t\}$
$\langle \mathsf{exec}, T_0 \rangle \models \mathsf{P}(t_1, \ldots, t_n)$	iff	$\mathsf{exec} = \mathsf{exec}'; \mathsf{P}(t_1, \ldots, t_n)$
$\langle \mathsf{exec}, T_0 \rangle \models \mathsf{C}(u)$	iff	$T_0 \vdash \mathsf{priv}(u)$ or $T_0 \vdash \mathsf{shk}(u, v)$ for some $v \neq \epsilon$
$\langle \mathsf{exec}, T_0 \rangle \models \Diamond\psi$	iff	$\exists i \in [0, \mathsf{length}(\mathsf{exec})]$ such that $\langle \mathsf{exec}_i, T_0 \rangle \models \psi$

Given a protocol Π and a set of ground terms T_0, we say that $\Pi \models \phi$ w.r.t. T_0, if $\langle \mathsf{exec}, T_0 \rangle \models \phi$ for all valid execution traces exec of Π w.r.t. T_0.

We now define the subset of $\mathcal{L}$ for which our result holds. We say a formula in $\mathcal{L}$ is *quantifier-free* if it does not contain any $\exists$. A formula is *modality-free* if it does not contain any $\Diamond$. We will only consider *attack formulas* of the form $\exists x_1. \ldots . \exists x_n.\phi'$ where ϕ' is quantifier-free, and we consider also some additional syntactic restrictions. Therefore, the security formulas we consider are of the form $\forall x_1, \ldots, \forall x_n.\neg\phi'$, i.e. the negation of an attack formula.

Definition 10 (attack formula)**.** An attack formula is an expression of the form

$$\exists x_1. \ldots . \exists x_n.\phi$$

where all the variables x_i are distinct and ϕ is a quantifier-free formula of $\mathcal{L}$ satisfying the following conditions:

(1) all subterms of ϕ are atomic terms with no names, i.e. $\mathsf{St}(\phi) \subseteq \mathcal{A} \cup \mathcal{X} \cup \mathcal{Y}$,
(2) for any term t, $\mathsf{learn}(t)$ can only occur positively in ϕ, i.e. under an even number of negations,
(3) any variable occurs at most once in a positive status event,
(4) if $\Diamond\psi$ is a subformula of ϕ that occurs negatively in ϕ, then a status event can only occur positively in ψ.

As we will see next this fragment is expressive enough to model classical security properties.

4.2. **Some security properties.** We now show how classical security properties like secrecy and several flavors of non-injective authentication properties can be expressed in our logic.

4.2.1. *Secrecy.* The secrecy property is the inability of the intruder to learn a message (e.g. a nonce, a key, or a compound term) that is specified (using a status event) as confidential. We will show how to specify the secrecy property for a nonce for example with a formula in $\mathcal{L}$. Let Π be a k-party protocol with

$$\Pi(j) = \lambda x_1^j. \ldots . \lambda x_k^j.\nu y_1^j. \ldots . \nu y_{p_j}^j.\mathsf{e}_1^j; \ldots ; \mathsf{e}_{\ell_j}^j \quad \text{for } 1 \leq j \leq k.$$

and let y_h^j ($1 \leq j \leq k$ and $1 \leq h \leq p_j$) be the nonce variable whose instantiations should remain confidential. In order to specify that all the instances of y_h^j must remain secret we build from Π, a protocol Π_{S} as follows. Let Secret be a predicate not occurring in Π, then

$$\Pi_{\mathsf{S}}(n) = \begin{cases} \Pi(n) & \text{for } 1 \leq n \leq k \text{ and } n \neq j \\ \lambda x_1^j. \ldots \lambda x_k^j. \nu y_1^j. \ldots \nu y_{p_j}^j. \mathsf{Secret}(x_1^j, \ldots, x_k^j, y_h^j); \mathsf{e}_1^j; \ldots; \mathsf{e}_{\ell_j}^j & \text{for } n = j \end{cases}$$

During an execution, the predicate Secret will link each instance $n_{y_h^j}^{sid}$ of y_h^j to the participants of the corresponding session *sid*. The following property expresses that the non-compromised instances of y_h^j should remain confidential

$$\phi_{\mathsf{S}} = \forall x_1. \ldots. \forall x_k. \forall y.\ [((\Diamond \mathsf{Secret}(x_1, \ldots, x_k, y)) \wedge \mathsf{NC}(x_1) \wedge \ldots \wedge \mathsf{NC}(x_k)) \Rightarrow \neg \mathsf{learn}(y)].$$

And the following formula is the corresponding attack formula

$$\overline{\phi_{\mathsf{S}}} = \exists x_1. \ldots. \exists x_k. \exists y.\ [(\Diamond \mathsf{Secret}(x_1, \ldots, x_k, y)) \wedge \mathsf{NC}(x_1) \wedge \ldots \wedge \mathsf{NC}(x_k) \wedge \mathsf{learn}(y)].$$

which satisfies the 4 conditions of the definition of an attack formula (Definition 10). Note that the same construction can be used to model the secrecy of a compound term t as seen by the agent executing the role $\Pi(j)$. For this, we simply add a status event $\mathsf{Secret}(x_1^j, \ldots, x_k^j, t)$ in $\Pi(j)$, and keep the attack formula unchanged. The 4 conditions stated in Definition 10 are still satisfied.

Example 6. Let us come back to the Needham-Schroeder protocol as presented in Example 3 to illustrate this property, and let's specify that the nonce y' generated by the responder is confidential. In order to do so, we build the 2-party protocol Π_{S} following the above mentioned construction, i.e. such that $\Pi_{\mathsf{S}}(1) = \Pi(1)$, and

$$\begin{aligned} \Pi_{\mathsf{S}}(2) = \ & \lambda x_A. \lambda x_B. \nu y'. \\ & \mathsf{Secret}(x_A, x_B, y') \\ & \mathsf{rcv}(\mathsf{enca}(\langle z', x_A \rangle, \mathsf{pub}(x_B))); \\ & \mathsf{snd}(\mathsf{enca}(\langle z', y' \rangle, \mathsf{pub}(x_A))); \\ & \mathsf{rcv}(\mathsf{enca}(y', \mathsf{pub}(x_B))) \end{aligned}$$

An attack on the secrecy of y', is any valid execution trace of Π_{S} that reveals to the intruder an honest instance of y' (i.e. generated by an honest session of $\Pi_{\mathsf{S}}(2)$). Formally, an attack on the secrecy of y' is a valid execution trace of Π_{S} that satisfies the following attack formula

$$\overline{\phi_{\mathsf{S}}} = \exists y_A. \exists y_B. \exists x.\ [(\Diamond \mathsf{Secret}(y_A, \ldots, y_B, x)) \wedge \mathsf{NC}(y_A) \wedge \mathsf{NC}(y_B) \wedge \mathsf{learn}(x)].$$

Let's consider as initial intruder knowledge $T_0 = \{a, b, c, \mathsf{priv}(c)\}$, the scenario $\mathsf{sc} = (1, s_1)(2, s_2)(2, s_2)(2, s_2)(1, s_1)(1, s_1)$, and the function α such that $\mathrm{dom}(\alpha) = \{s_1, s_2\}$, $\alpha(s_1) = (a, c)$, and $\alpha(s_2) = (a, b)$. We denote by tr the symbolic trace associated to Π_{S}, sc, and α. Let σ be the substitution such that $\sigma = \{z^{s_1} \mapsto n_{y'}^{s_2},\ z'^{s_2} \mapsto n_y^{s_1}\}$. The execution trace $\mathsf{tr}\sigma$ is valid w.r.t. T_0. This execution corresponds to the famous attack due to Lowe [24], and formally satisfies $\overline{\phi_{\mathsf{S}}}$, i.e. $\langle T_0, \mathsf{tr}\sigma \rangle \models \overline{\phi_{\mathsf{S}}}$, and thus $\Pi_{\mathsf{S}} \not\models \phi_{\mathsf{S}}$ w.r.t. T_0.

We are now going to look at how to formally express authentication properties.

4.2.2. *Aliveness.* We start with the weakest notion of authentication in the hierarchy of Lowe [25], namely aliveness. Informally, a protocol Π satisfies aliveness if and only if each time a participant a finishes an honest session involving participant b (of any of the roles of Π), b has at least partially executed one session (of any of the roles of Π), and in that sense b is alive.

In order to express this property, we need to detect in the executions of Π, each time a session starts and ends. This can be achieved by adding status events at the beginning and the end of each role. More precisely, if we consider the k-party protocol Π with

$$\Pi(j) = \lambda x_1^j. \ldots \lambda x_k^j. \nu y_1^j. \ldots \nu y_{p_j}^j. \mathsf{e}_1^j; \ldots; \mathsf{e}_{\ell_j}^j \quad \text{for } 1 \leq j \leq k.$$

We build the protocol Π_{A} by inserting new status events as follows:

$$\Pi_{\mathsf{A}}(j) = \lambda x_1^j. \ldots \lambda x_k^j. \nu y_1^j. \ldots \nu y_{p_j}^j. \mathsf{Start}(x_j^j); \mathsf{e}_1^j; \ldots; \mathsf{e}_{\ell_j}^j; \mathsf{End}(x_1^j, \ldots, x_k^j) \quad \text{for } 1 \leq j \leq k.$$

where the predicates Start and End will mark in an execution the beginning and the end of each session, and will link together the effective participants of each session. Aliveness can then be modelled by the following formula

$$\phi_{\mathsf{A}} = \left\{ \begin{array}{l} \forall y_1. \ldots \forall y_k. \big[\mathsf{End}(y_1, \ldots, y_k) \ \wedge \ \mathsf{NC}(y_1) \ \wedge \ \ldots \ \wedge \ \mathsf{NC}(y_k) \\ \qquad\qquad\qquad\qquad\qquad\qquad \Rightarrow \ \Diamond\mathsf{Start}(y_1) \wedge \ldots \wedge \Diamond\mathsf{Start}(y_k)\big] \end{array} \right.$$

An attack on protocol Π w.r.t. aliveness is thus a trace of Π_{A} satisfying the following attack formula

$$\overline{\phi_{\mathsf{A}}} = \left\{ \begin{array}{l} \exists y_1. \ldots \exists y_k. \big[\mathsf{End}(y_1, \ldots, y_k) \ \wedge \ \mathsf{NC}(y_1) \ \wedge \ \ldots \ \wedge \ \mathsf{NC}(y_k) \\ \qquad\qquad\qquad\qquad\qquad\qquad \wedge \ \big(\neg\Diamond\mathsf{Start}(y_1) \vee \ldots \vee \neg\Diamond\mathsf{Start}(y_k)\big)\big] \end{array} \right.$$

Example 7. Let us come back to the Needham-Schroeder protocol as presented in Example 3 to illustrate this property. In order to do so, we build the 2-party protocol Π_{A} following the above mentioned construction, i.e. such that

$$\begin{array}{lll} \Pi_{\mathsf{A}}(1) = & \lambda x_A. \lambda x_B. \nu y. & \\ & \mathsf{Start}(x_A) & \\ & \mathsf{snd}(\mathsf{enca}(\langle y, x_A\rangle, \mathsf{pub}(x_B))); & \\ & \mathsf{rcv}(\mathsf{enca}(\langle y, z\rangle, \mathsf{pub}(x_A))); & \\ & \mathsf{snd}(\mathsf{enca}(z, \mathsf{pub}(x_B))) & \\ & \mathsf{End}(x_A, x_B) & \end{array} \qquad \begin{array}{lll} \Pi_{\mathsf{A}}(2) = & \lambda x_A. \lambda x_B. \nu y'. & \\ & \mathsf{Start}(x_B) & \\ & \mathsf{rcv}(\mathsf{enca}(\langle z', x_A\rangle, \mathsf{pub}(x_B))); & \\ & \mathsf{snd}(\mathsf{enca}(\langle z', y'\rangle, \mathsf{pub}(x_A))); & \\ & \mathsf{rcv}(\mathsf{enca}(y', \mathsf{pub}(x_B))) & \\ & \mathsf{End}(x_A, x_B) & \end{array}$$

Now this protocol satisfies aliveness, if in every valid execution trace of Π_{A} during which an agent a executing an honest session of role $\Pi_{\mathsf{A}}(1)$ (*resp.* $\Pi_{\mathsf{A}}(2)$) with agent b, b has also initiated a session of the protocol. Formally, Π satisfies aliveness if every valid execution trace of Π_{A} satisfies the following formula;

$$\phi_{\mathsf{A}} = \ \forall x_A. \forall x_B. \ \mathsf{End}(x_A, x_B) \ \wedge \ \mathsf{NC}(x_A) \ \wedge \ \mathsf{NC}(x_B) \ \Rightarrow \ [\Diamond\mathsf{Start}(x_A) \wedge \Diamond\mathsf{Start}(x_B)]$$

Consider the symbolic trace tr associated to the scenario

$$\mathsf{sc}_{\mathsf{A}} = (1, s_1)(1, s_1)(2, s_2)(2, s_2)(2, s_2)(1, s_1)(1, s_1)(1, s_1)(2, s_2)(2, s_2)$$

and the function α as defined in Example 6. Actually, we have that $\langle T_0, \mathsf{tr}\sigma\rangle \models \phi_{\mathsf{A}}$ using the set T_0 and the substitution σ as defined in Example 6. More generally, using an automatic tool such as ProVerif [8], one can prove that the Needham-Schroeder protocol satisfies aliveness w.r.t. the initial intruder knowledge $T_0 = \{a, b, c, \mathsf{priv}(c)\}$, i.e. $\Pi_{\mathsf{A}} \models \phi_{\mathsf{A}}$ w.r.t. T_0.

4.2.3. *Weak agreement.* Weak agreement is slightly stronger than aliveness. Informally, a protocol Π satisfies weak agreement, if and only if each time a participant a finishes an honest session involving participant b (of any of the roles of Π), b has at least initiated a session involving a (of any of the roles of Π).

Again, in order to express this property, we need to detect in the executions of Π, each time a session starts and ends, but also which participants are involved in each session that is initiated. This can be achieved by adding status events at the beginning and the end of each role. More precisely, if we consider the k-party protocol Π with

$$\Pi(j) = \lambda x_1^j. \ldots \lambda x_k^j. \nu y_1^j. \ldots \nu y_{p_j}^j. \mathsf{e}_1^j; \ldots; \mathsf{e}_{\ell_j}^j \quad \text{for } 1 \leq j \leq k.$$

We build the protocol Π_{WA} by inserting new status events as follows:

$$\Pi_{\mathsf{WA}}(j) = \lambda x_1^j. \ldots \lambda x_k^j. \nu y_1^j. \ldots \nu y_{p_j}^j. \mathsf{Start}(x_j^j, x_1^j); \ldots; \mathsf{Start}_{j,k}(x_j^j, x_k^j); \\ \mathsf{e}_1^j; \ldots; \mathsf{e}_{\ell_j}^j; \mathsf{End}_j(x_1^j, \ldots, x_k^j) \qquad \text{for } 1 \leq j \leq k.$$

where the predicates Start and End will mark in an execution the beginning and the end of each session, and will link together the effective participants of each session both at the beginning and the end of the session. Weak agreement can then be modelled by the following formula

$$\phi_{\mathsf{WA}} = \forall y_1^1. \ldots \forall y_k^1. \ldots \forall y_1^k. \ldots \forall y_k^k.$$
$$\bigwedge_{j \in \{1,\ldots,k\}} \left[\mathsf{End}_j(y_1^j, \ldots, y_k^j) \ \wedge \ \mathsf{NC}(y_1^j) \ \wedge \ \ldots \ \wedge \ \mathsf{NC}(y_k^j) \ \Rightarrow \bigwedge_{i \in \{1,\ldots,k\}, i \neq j} \Diamond \mathsf{Start}(y_i^j, y_j^j) \right]$$

An attack on protocol Π w.r.t. aliveness is thus a trace of Π_{WA} satisfying the following attack formula

$$\overline{\phi_{\mathsf{WA}}} \equiv \exists y_1. \ldots \exists y_k. \ \mathsf{End}_j(y_1, \ldots, y_k) \ \wedge \ \mathsf{NC}(y_1) \ \wedge \ \ldots \ \wedge \ \mathsf{NC}(y_k) \ \wedge \ \neg \Diamond \mathsf{Start}(y_i, y_j)$$

for some $j, i \in \{1, \ldots, k\}$ with $i \neq j$.

Example 8. Let us come back to the Needham-Schroeder protocol as presented in Example 3 to illustrate this property. In order to do so, we build the 2-party protocol Π_{WA} following the above mentioned construction, i.e. such that:

$$\begin{array}{lll} \Pi_{\mathsf{WA}}(1) = & \lambda x_A. \lambda x_B. \nu y. & \Pi_{\mathsf{WA}}(2) = \quad \lambda x_A. \lambda x_B. \nu y'. \\ & \mathsf{Start}(x_A, x_A) & \mathsf{Start}(x_B, x_A) \\ & \mathsf{Start}(x_A, x_B) & \mathsf{Start}(x_B, x_B) \\ & \mathsf{snd}(\mathsf{enca}(\langle y, x_A \rangle, \mathsf{pub}(x_B))); & \mathsf{rcv}(\mathsf{enca}(\langle z', x_A \rangle, \mathsf{pub}(x_B))); \\ & \mathsf{rcv}(\mathsf{enca}(\langle y, z \rangle, \mathsf{pub}(x_A))); & \mathsf{snd}(\mathsf{enca}(\langle z', y' \rangle, \mathsf{pub}(x_A))); \\ & \mathsf{snd}(\mathsf{enca}(z, \mathsf{pub}(x_B))) & \mathsf{rcv}(\mathsf{enca}(y', \mathsf{pub}(x_B))) \\ & \mathsf{End}_1(x_A, x_B) & \mathsf{End}_2(x_A, x_B) \end{array}$$

Now this protocol satisfies weak agreement, if in every valid execution trace of Π_{WA} during which an agent a executing an honest session of role $\Pi_{\mathsf{WA}}(1)$ (*resp.* $\Pi_{\mathsf{WA}}(2)$) with agent b, b has also initiated a session of the protocol involving agent a. In other words, Π admits an attack w.r.t. weak agreement if there exists a valid execution trace of Π_{WA} that

satisfies the following formula:

$$\overline{\phi_{\mathsf{WA}}} \equiv \exists x_A^1.\exists x_B^1.\exists x_A^2.\exists x_B^2.$$
$$\left[\begin{array}{l} \mathsf{End}_1(x_A^1,x_B^1) \ \wedge\ \mathsf{NC}(x_A^1) \ \wedge\ \mathsf{NC}(x_B^1) \ \wedge\ \neg\Diamond\mathsf{Start}(x_B^1,x_A^1) \\ \qquad\qquad\qquad\qquad \vee \\ \mathsf{End}_2(x_A^2,x_B^2) \ \wedge\ \mathsf{NC}(x_A^2) \ \wedge\ \mathsf{NC}(x_B^2) \ \wedge\ \neg\Diamond\mathsf{Start}(x_A^2,x_B^2) \end{array}\right]$$

Let's consider as initial intruder knowledge $T_0 = \{a, b, c, \mathsf{priv}(c)\}$, the scenario

$$\mathsf{sc} = (1,s_1)(1,s_1)(1,s_1)(2,s_2)(2,s_2)(2,s_2)(2,s_2)(1,s_1)(1,s_1)(1,s_1)(2,s_2)(2,s_2)(2,s_2),$$

the function α such that $\mathrm{dom}(\alpha) = \{s_1, s_2\}$, $\alpha(s_1) = (a, c)$, and $\alpha(s_2) = (a, b)$, and the substitution $\sigma = \{z^{s_1} \mapsto n_{y'}^{s_2},\ z'^{s_2} \mapsto n_y^{s_1}\}$. The execution trace $\mathsf{tr}\sigma$ is valid w.r.t. T_0 with tr the symbolic trace associated to sc and α. This execution corresponds to the famous attack due to Lowe [24], and formally satisfies $\overline{\phi_{\mathsf{WA}}}$, i.e. $\langle T_0, \mathsf{tr}\sigma\rangle \models \overline{\phi_{\mathsf{WA}}}$, and thus $\Pi_{\mathsf{WA}} \not\models \phi_{\mathsf{WA}}$ w.r.t. T_0.

5. Transformation of protocols

In Section 5.1 we define our transformation before we state our main result in Section 5.2 whose proof is postponed to Part II.

5.1. **Our transformation.** Given an input protocol Π, our transformation will compute a new protocol $\widetilde{\Pi}$ which consists in two phases. During the first phase, the protocol participants try to agree on some common, dynamically generated, session identifier τ. For this, each participant sends a freshly generated nonce N_i together with his identity A_i to all other participants. (Note that if broadcast is not practical or if not all identities are known to each participant, the message can be sent to some of the participants who forwards the message.) At the end of this preamble, each participant computes a session identifier: $\tau = \langle\langle A_1, N_1\rangle, \ldots, \langle A_k, N_k\rangle\rangle$. Note that an active attacker may interfere with this initialization phase and may intercept and replace some of the nonces. Hence, the protocol participants do not necessarily agree on the same session identifier τ after this preamble. In fact, each participant computes his own session identifier, say τ_j. During the second phase, each participant j executes the original protocol in which the dynamically computed identifier is used for tagging each application of a cryptographic primitive. In this phase, when a participant opens an encryption, he checks that the tag is in accordance with the nonces he received during the initialization phase. In particular, he can test the presence of his own nonce.

The transformation, using the informal Alice-Bob notation, is described below and relies on the tagging operation that is formally defined in Definition 11.

$$\Pi = \left\{\begin{array}{ll} A_{i_1} \to A_{j_1} : & m_1 \\ \vdots & \\ A_{i_\ell} \to A_{j_\ell} : & m_\ell \end{array}\right. \qquad \widetilde{\Pi} = \left\{\begin{array}{lll} \text{Phase 1} & & \text{Phase 2} \\ A_1 \to All : & \langle A_1, N_1\rangle & A_{i_1} \to A_{j_1} : \ [m_1]_\tau \\ \vdots & & \vdots \\ A_k \to All : & \langle A_k, N_k\rangle & A_{i_\ell} \to A_{j_\ell} : \ [m_\ell]_\tau \\ \multicolumn{3}{l}{\text{where } \tau = \langle \mathsf{tag}_1, \ldots, \mathsf{tag}_k\rangle \text{ with } \mathsf{tag}_i = \langle A_i, N_i\rangle} \end{array}\right.$$

Note that, the Alice-Bob notation only represents what happens in a normal execution, i.e. with no intervention of the attacker. Of course, in such a situation, the participants agree on the same session identifier τ used in the second phase.

Definition 11 (k-tag, k-tagging)**.** A *k-tag* is a term $\langle\langle a_1, v_1\rangle, \ldots, \langle a_k, v_k\rangle\rangle$ where each $a_i \in \mathcal{A}$ and each v_i is a term. Let u be a term and tag be a k-tag. The k-tagging of u with tag, denoted $[u]_{\mathsf{tag}}$, is inductively defined as follows:

$$\begin{array}{lcll} [\langle u_1, u_2\rangle]_{\mathsf{tag}} & = & \langle [u_1]_{\mathsf{tag}}, [u_2]_{\mathsf{tag}}\rangle & \\ [\mathsf{f}(u_1, u_2)]_{\mathsf{tag}} & = & \mathsf{f}(\langle \mathsf{tag}, [u_1]_{\mathsf{tag}}\rangle, [u_2]_{\mathsf{tag}}) & \text{for } \mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}\} \\ [\mathsf{h}(u_1)]_{\mathsf{tag}} & = & \mathsf{h}(\langle \mathsf{tag}, [u_1]_{\mathsf{tag}}\rangle) & \\ [u]_{\mathsf{tag}} & = & u & \text{otherwise} \end{array}$$

We say that a term t is *k-tagged* if $u|_{1.1}$ is a k-tag for any $u \in \mathsf{CryptSt}(t)$.

These notions are extended to events and sequences of events as expected. We are now able to formally define our transformation.

Definition 12 (protocol transformation)**.** Let Π be a k-party protocol such that

$$\Pi(j) = \lambda x_1^j \ldots \lambda x_k^j . \nu y_1^j \ldots \nu y_{p_j}^j . \mathsf{seq}^j \quad \text{for } 1 \leq j \leq k.$$

and the variables z_i^j $(1 \leq i, j \leq k)$ do not appear in Π (which can always be ensured by renaming variables in Π). The transformed protocol $\widetilde{\Pi}$ is a k-party protocol defined as follows:

$$\widetilde{\Pi}(j) = \lambda x_1^j \ldots \lambda x_k^j . \nu y_1^j \ldots \nu y_{p_j}^j . \nu z_j^j . \widetilde{\Pi}^{\mathsf{init}}(j); [\mathsf{seq}^j]_{\tau_j} \quad \text{for } 1 \leq j \leq k$$

where

$$\widetilde{\Pi}^{\mathsf{init}}(j) = \mathsf{rcv}(u_1^j); \ldots; \mathsf{rcv}(u_{j-1}^j); \mathsf{snd}(u_j^j); \mathsf{rcv}(u_{j+1}^j); \ldots; \mathsf{rcv}(u_k^j)$$

and $\tau_j = \langle u_1^j, \ldots, u_k^j\rangle$ with $u_i^j = \langle x_i^j, z_i^j\rangle$.

In the above definition, the protocol $\widetilde{\Pi}^{\mathsf{init}}$ models the initialization phase and the variables z_i^j correspond to the nonces that are generated and exchanged during this phase. In particular for the role j, the variable z_j^j is a freshly generated nonce while the other variables z_i^j, $i \neq j$, are expected to be bound to the other participant's nonces in the receive events. Remember also that the variables x_i^j are the role parameters which correspond to the agents. The tag computed by the j^{th} role in our transformation consists in the concatenation of the k names of the agents involved in the protocol, together with the $k-1$ terms received during the initialization phase as well as the fresh nonce generated by the role j itself, i.e. z_j^j. We illustrate this transformation on the Needham-Schroeder protocol introduced in Section 2.

Example 9. Consider the Needham-Schroeder protocol described in Example 3. Applying our transformation we obtain a 2-party protocol $\widetilde{\Pi}$. The role $\widetilde{\Pi}(2)$ is described below. The role $\widetilde{\Pi}(1)$ can be obtained in a similar way.

$$\begin{array}{rl} \widetilde{\Pi}(2) = & \lambda x_A \lambda x_B . \nu y' . \nu z_B . \mathsf{rcv}(\langle x_A, z_A\rangle); \mathsf{snd}(\langle x_B, z_B\rangle); \\ & \mathsf{rcv}(\mathsf{enca}(\langle \tau, \langle z', x_A\rangle\rangle, \mathsf{pub}(x_B))); \\ & \mathsf{snd}(\mathsf{enca}(\langle \tau, \langle z', y'\rangle\rangle, \mathsf{pub}(x_A))); \\ & \mathsf{rcv}(\mathsf{enca}(\langle \tau, y'\rangle, \mathsf{pub}(x_B))) \end{array}$$

where $\tau = \langle\langle x_A, z_A\rangle, \langle x_B, z_B\rangle\rangle$. Note that Lowe's famous man-in-the-middle attack [24] described in Example 8 does not exist anymore on $\widetilde{\Pi}$.

5.2. **Main theorem.** Roughly, our result states that if the compiled protocol admits an attack that may involve several sessions, then there exists an attack which only requires a bounded number of sessions of each role, and the bound only depends on the security formula under study. More formally, we define the size of a formula as follows:

Definition 13 (size of a formula)**.** Let ϕ be a formula. The *size* of ϕ, denoted $\|\phi\|$, is defined as follows:

$$\begin{array}{rcl} \|\mathsf{true}\| & \stackrel{\mathsf{def}}{=} & 0 \\ \|\mathsf{P}(t_1,\ldots,t_n)\| & \stackrel{\mathsf{def}}{=} & 1 \\ \|\mathsf{learn}(t)\| & \stackrel{\mathsf{def}}{=} & 0 \\ \|\mathsf{C}(t)\| & \stackrel{\mathsf{def}}{=} & 0 \\ \|\neg\phi\| & \stackrel{\mathsf{def}}{=} & \|\phi\|^- \\ \|\phi_1 \vee \phi_2\| & \stackrel{\mathsf{def}}{=} & \max\{\|\phi_1\|, \|\phi_2\|\} \\ \|\exists x.\ \phi\| & \stackrel{\mathsf{def}}{=} & \|\phi\| \\ \|\Diamond\phi\| & \stackrel{\mathsf{def}}{=} & \|\phi\| \end{array} \qquad \begin{array}{rcl} \|\mathsf{true}\|^- & \stackrel{\mathsf{def}}{=} & 0 \\ \|\mathsf{P}(t_1,\ldots,t_n)\|^- & \stackrel{\mathsf{def}}{=} & 1 \\ \|\mathsf{learn}(t)\|^- & \stackrel{\mathsf{def}}{=} & 0 \\ \|\mathsf{C}(t)\|^- & \stackrel{\mathsf{def}}{=} & 0 \\ \|\neg\phi\|^- & \stackrel{\mathsf{def}}{=} & \|\phi\| \\ \|\phi_1 \vee \phi_2\|^- & \stackrel{\mathsf{def}}{=} & \|\phi_1\|^- + \|\phi_2\|^- \\ \|\exists x.\ \phi\|^- & \stackrel{\mathsf{def}}{=} & \|\phi\|^- \\ \|\Diamond\phi\|^- & \stackrel{\mathsf{def}}{=} & 0 \end{array}$$

Intuitively, when an attack trace involves several sessions of each role, not all the sessions are necessary to mount the attack. We only need to keep those sessions that witness the satisfiability of the attack formula ϕ. By definition of an attack formula (see Definition 10), we know that each variable occurring in ϕ also occurs in a positive status events. Thus, there is no need to take into account the number of occurrences of $\mathsf{learn}(t)$ in the previous definition.

Example 10. Note that $\|\phi_1 \wedge \phi_2\| = \|\phi_1\| + \|\phi_2\|$. Considering the attack formulas $\overline{\phi_{\mathsf{S}}}$, $\overline{\phi_{\mathsf{A}}}$, and $\overline{\phi_{\mathsf{WA}}}$ as defined in Section 4.2, we have that $\|\overline{\phi_{\mathsf{S}}}\| = \|\overline{\phi_{\mathsf{A}}}\| = \|\overline{\phi_{\mathsf{WA}}}\| = 1$.

We are now able to state our main transference result.

Theorem 1. Let Π be a k-party protocol, $\widetilde{\Pi}$ be its corresponding transformed protocol and T_0 be a set of ground atoms such that $\mathit{lgKeys}(\Pi) \cap \mathit{plaintext}(\Pi) \subseteq T_0 \cup \mathcal{K}_\epsilon$. Let ϕ be an attack formula such that $\widetilde{\Pi} \models \phi$ w.r.t. T_0. There exists a valid execution trace exec of $\widetilde{\Pi}$ such that:

$$\langle \mathsf{exec}, T_0 \rangle \models \phi \text{ and } \mathsf{exec} \text{ involves at most } \|\phi\| \text{ sessions of each role.}$$

Applying our result, we can now establish that if a protocol built according to our transformation admits an attack on secrecy (resp. aliveness, weak agreement), then it admits an attack that involves at most one session of each role. The situation is however slightly more complicated than it may seem at first sight. As we have an infinite number of agent names there is an infinite number of sessions, which one would need to verify separately. Actually we can avoid this combinatorial explosion thanks to the following well-known result [14]: when verifying secrecy properties it is sufficient to consider two agents (an honest agent and a dishonest one). Hence, using this result, we can instantiate all the parameters using only two agent names. Similar reduction results also exist for authentication properties (see [14]).

Note that we only consider protocols whose long-term secret keys do not occur in plaintext position. This assumption is required to ensure that the "small scenario", (*i.e.*, the one that involves only $\|\phi\|$ sessions of each role) will violate the same security property ϕ.

We may actually relax this assumption if we consider an execution trace that reveals such a long-term key as a violation of the security property as well. The result is stated in this way in [1].

Actually, for the security properties presented in the previous section, we can go even further and only consider one *honest* session of each role.

Corollary 1. Let Π be a k-party protocol, Π_{S} (respectively, Π_{A}, Π_{WA}) be the annotated protocol for modeling secrecy (respectively aliveness and weak agreement) as defined in Section 4.2.1, and $\widetilde{\Pi}_{\mathsf{S}}$ (respectively, $\widetilde{\Pi}_{\mathsf{A}}$, $\widetilde{\Pi}_{\mathsf{WA}}$) the corresponding transformed protocol. Let T_0 be a set of ground atoms such that $\mathit{lgKeys}(\Pi) \cap \mathit{plaintext}(\Pi) \subseteq T_0 \cup \mathcal{K}_\epsilon$ and $\overline{\phi_{\mathsf{S}}}$ (respectively $\overline{\phi_{\mathsf{A}}}$, $\overline{\phi_{\mathsf{WA}}}$) an attack formula against secrecy (respectively aliveness and weak agreement) as defined in Section 4.2.1. For $\mathsf{X} \in \{\mathsf{S}, \mathsf{A}, \mathsf{WA}\}$ we have that if $\widetilde{\Pi}_{\mathsf{X}} \models \overline{\phi_{\mathsf{X}}}$ w.r.t. T_0 then there exists a valid execution trace exec of $\widetilde{\Pi}_{\mathsf{X}}$ such that:

$$\langle \mathsf{exec}, T_0 \rangle \models \overline{\phi_{\mathsf{X}}} \text{ and } \mathsf{exec} \text{ involves at most one } \textit{honest} \text{ session of each role.}$$

5.3. **Alternative ways of tagging protocols.** Our transformation is computationally light as it does not add any cryptographic application. However, it increases significantly the size of messages to be encrypted and signed. As an alternative, we may choose to hash the tags. Our results still hold in this setting.

We have also considered an alternative, slightly different transformation that does not include the identities in the tag, i.e., the tag is simply the sequence of nonces. Our main result, Theorem 1, still holds as the proof does not use the presence of the identities. However, the stronger results presented on particular properties stated in Corollary 1, do not hold anymore, as the proof crucially relies on the presence of the agent names in the tag. When omitting identities, even for secrecy, we need to additionally check for attacks that involve a session engaged with the attacker. Indeed, on the example of the Needham-Schroeder protocol the man-in-the-middle attack is not prevented by this weaker tagging scheme. However, the result requires one to also consider one dishonest session for each role, hence including the attack scenario. In both cases, it is important for the tags to be *collaborative*, i.e. all participants do contribute by adding a fresh nonce.

— PART II: Proof of our reduction result —

In this part, we give an overview of the proof of our reduction result stated in Theorem 1. Assume that our protocol $\widetilde{\Pi}$ admits an attack.

(1) We first show that there is an attack on a *well-formed* execution trace (Section 6). In a well-formed execution trace (see Definition 17), terms are necessarily tagged with the expected tag, i.e. the tag computed during the initialization phase. Moreover, only names coming from sessions tagged in the same way can be used in the events of those sessions. In order to prove this, we define a transformation $\overline{\cdot}$ that transforms an execution trace to a well-formed one by abstracting some subterms (those that are not tagged properly using the expected tag) by fresh nonces. We show that this transformation preserves the validity of the trace (Proposition 2) as well as the satisfiability of the attack formula under study (Proposition 3).

(2) Then, given a set of sessions S and a valid and well-formed execution exec that satisfies the attack formula, we show that $\mathsf{exec}|_S$, i.e. the restriction of exec to the events coming from a session in S is still an execution satisfying the attack formula. Since messages coming from one session can be used to build a message for another session, this can only be achieved by requiring some conditions on S. Basically, to ensure the validity of the execution $\mathsf{exec}|_S$, we have to ensure that sessions that share the same tag are either all in S or none of them is in S (see Proposition 4). Then, to ensure the satisfiability of the attack formula, we have to keep enough sessions but we can bound a priori the number of sessions that is needed to mount an attack (see Proposition 5).

6. First step: towards a well-formed execution trace

In this section, we formally define our notion of *well-formedness* and we propose a transformation that allows us to transform a trace exec into a well-formed one $\overline{\mathsf{exec}}$ (Section 6.2) preserving its validity (Section 6.3) and the satisfiability of the attack formula (Section 6.4).

6.1. **Well-formed.** The idea behind our notion of well-formedness is to ensure that each term will be properly tagged. Basically, this means that each term has to be tagged with its expected tag, i.e. the one computed during the initialization phase of the protocol (phase 1). From now on, when we consider a trace exec issued from a protocol $\widetilde{\Pi}$, we assume that the events occurring in exec are annotated with their session identifier, and we write $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ when we want to refer to these annotations explicitly.

The transformation that we consider will abstract some subterms by fresh names from the intruder's knowledge (*i.e.* names in $\mathcal{N}_\epsilon$). Those names will be denoted by $n_t^{\epsilon,S}$ where S is set of session identifiers, and t is a term. Intuitively, such a name will be used to abstract the subterm t when used in an event from a session $sid \in S$. We assume that those names (which constitute an infinite subset of $\mathcal{N}_\epsilon$) are not used anywhere else. In particular, they do not occur in the execution trace before applying our transformation.

Definition 14 ($\mathsf{ExpectedTag}(\mathsf{exec}, sid)$)**.** Let Π be a k-party protocol and let $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be an execution trace of $\widetilde{\Pi}$. Let sid be a session identifier and $[\mathsf{e}_{i_1}^{sid}; \ldots; \mathsf{e}_{i_h}^{sid}]$

(with $1 \leq i_1 < \ldots < i_h \leq \ell$) be the sequence of communication events in exec that are annotated with sid. We define the *expected tag of a session sid in* exec as

- $\mathsf{ExpectedTag}(\mathsf{exec}, sid) = \bot$ when $h < k$,
- $\mathsf{ExpectedTag}(\mathsf{exec}, sid) = \langle m_1, \ldots, m_k \rangle$ otherwise, where for all $j \in \{1, \ldots, k\}$, m_j is such that $\mathsf{e}_{i_j}^{sid} = \mathsf{rcv}(m_j)$ or $\mathsf{e}_{i_j}^{sid} = \mathsf{snd}(m_j)$.

Roughly, the expected tag associated to a session sid is the one obtained by putting together the messages that occur in the k first communication events annotated with sid that occur in exec. When those events do not exist, the expected tag of sid is undefined. We define $\mathsf{ExpectedTags}(\mathsf{exec})$ to denote the set of expected tags that occur in the trace exec. More formally, we have that:

$$\mathsf{ExpectedTags}(\mathsf{exec}) = \bigcup_{sid} \{\mathsf{ExpectedTag}(\mathsf{exec}, sid)\}.$$

Since a session is the execution of one role, it is likely that several sessions will have the same expected tag. However, note that sessions that correspond to the execution of the same role (e.g. the j^{th} role) cannot have the same expected tag since the tag will contain a fresh nonce at its j^{th} position.

Definition 15 ($\mathsf{sameTagAs}(\mathsf{exec}, sid)$)**.** Let Π be a k-party protocol and let exec be an execution trace (not necessarily valid) of $\widetilde{\Pi}$. We define $\mathsf{sameTagAs}(\mathsf{exec}, sid)$ to be the set of sessions sharing the same expected tag with the session sid, i.e.

$$\mathsf{sameTagAs}(\mathsf{exec}, sid) = \begin{cases} \{sid\} & \text{if } \mathsf{ExpectedTag}(\mathsf{exec}, sid) = \bot \\ \{sid' \mid \mathsf{ExpectedTag}(\mathsf{exec}, sid') = \mathsf{ExpectedTag}(\mathsf{exec}, sid)\} & \text{otherwise} \end{cases}$$

Our notion of well-formedness aims to ensure that each event that occurs in a trace is tagged properly. For this, we first define $\mathsf{Tags}(\mathsf{exec}, sid)$. This set corresponds to the tags that actually occur in the events issued from the session sid in the execution trace exec.

Definition 16 ($\mathsf{Tags}(\mathsf{exec}, sid)$)**.** Let Π be a k-party protocol and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be an execution trace of $\widetilde{\Pi}$ which is k-tagged. Let sid be a session identifier. We define the tags of a session sid in exec as follows:

$$\mathsf{Tags}(\mathsf{exec}, sid) = \{u|_{1.1} \mid u \in \mathsf{CryptSt}(\mathsf{e}_j^{sid_j}) \text{ for some } j \in \{1, \ldots, \ell\} \text{ such that } sid_j = sid\}.$$

We define $\mathsf{Tags}(\mathsf{exec})$ to denote the set of tags that occur in the trace exec. More formally, we have that

$$\mathsf{Tags}(\mathsf{exec}) = \bigcup_{sid} \mathsf{Tags}(\mathsf{exec}, sid).$$

We are now able to define our notion of well-formed execution trace.

Definition 17 (well-formed execution trace)**.** Let Π be a k-party protocol, and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be an execution trace associated to $\widetilde{\Pi}$. We say that exec is *well-formed* if:

(1) exec is k-tagged, i.e. for all $t \in \mathsf{St}(\mathsf{exec})$, t is k-tagged;
(2) $\mathsf{Tags}(\mathsf{exec}, sid) \subseteq \{\mathsf{ExpectedTag}(\mathsf{exec}, sid)\}$ for every sid;
(3) For every i, we have that $names(\mathsf{e}_i^{sid_i}) \subseteq \{n_t^{\epsilon,S} \mid t \in T\} \cup \{n_y^{sid} \mid y \in \mathcal{Y} \text{ and } sid \in S\}$ where $S = \mathsf{sameTagAs}(\mathsf{exec}, sid_i)$.

Intuitively, in a well-formed trace, the events of a session *sid* are k-tagged with the expected tag, i.e. the tag defined in the preamble of the session *sid*. Moreover, the nonces used in a session *sid* are those that are generated in a session that used the same tag as *sid* (or they come from the intruder).

6.2. **Our transformation of execution traces.** A valid execution trace is not necessarily well-formed. Our goal is to show that we can however always transform an execution trace into a well-formed execution trace. The main idea is to replace each subterm that is not tagged in the expected way with a nonce known by the attacker. The difficulty will be to ensure that the resulting trace is still a valid one (see Section 6.3) and still a witness of the existence of an attack (see Section 6.4).

We first define our transformation on a term. For this we need to introduce the notion of $\mathsf{HeadTag}$

Definition 18 ($\mathsf{HeadTag}(\mathsf{exec}, t)$)**.** Let Π be a k-party protocol and exec be an execution trace (not necessarily valid) of $\widetilde{\Pi}$. We define the head tag of a term t w.r.t. the trace exec, denoted $\mathsf{HeadTag}(\mathsf{exec}, t)$.

$$\mathsf{HeadTag}(\mathsf{exec}, t) = \begin{cases} \tau & \text{if } t = \mathsf{f}(\langle \tau, u\rangle, u_2, \ldots, u_n) \in \mathsf{CryptSt}(t) \\ & \quad \text{and } \tau \in \mathsf{ExpectedTags}(\mathsf{exec}) \\ \bot & \text{otherwise} \end{cases}$$

Roughly, our transformation of a term proceeds as follows. We replace each cryptographic subterm which is not tagged properly with a nonce. We also perform the same kind of replacement on nonces to ensure that sessions that are tagged differently will not share any nonces.

Definition 19 ($\overline{t}^{\mathsf{exec},sid}$)**.** Let Π be a k-party protocol, exec be an execution trace (not necessarily valid) of $\widetilde{\Pi}$, *sid* be a session identifier and $\tau = \mathsf{ExpectedTag}(\mathsf{exec}, sid)$.

- $\overline{n}^{\mathsf{exec},sid} = n_n^{\epsilon,S}$ if $n \in \mathcal{N}^\epsilon$ or if $\tau = \bot$, where $S = \mathsf{sameTagAs}(\mathsf{exec}, sid)$;
- $\overline{n_y^{sid'}}^{\mathsf{exec},sid} = \begin{cases} n_y^{sid'} \text{ if } sid' \in \mathsf{sameTagAs}(\mathsf{exec}, sid) \\ n_{n_y^{sid'}}^{\epsilon,S} \text{ where } S = \mathsf{sameTagAs}(\mathsf{exec}, sid) \text{ otherwise ;} \end{cases}$
- $\overline{a}^{\mathsf{exec},sid} = a$ if a is the name of an agent;
- $\overline{\mathsf{f}(a_1, \ldots, a_n)}^{\mathsf{exec},sid} = \mathsf{f}(a_1, \ldots, a_n)$ for $\mathsf{f} \in \{\mathsf{shk}, \mathsf{pub}, \mathsf{priv}\}$
- $\overline{\langle u, v\rangle}^{\mathsf{exec},sid} = \langle \overline{u}^{\mathsf{exec},sid}, \overline{v}^{\mathsf{exec},sid}\rangle$;
- $\overline{\mathsf{f}(u_1, \ldots, u_n)}^{\mathsf{exec},sid} = \begin{cases} \mathsf{f}(\overline{u_1}^{\mathsf{exec},sid}, \ldots, \overline{u_n}^{\mathsf{exec},sid}) \\ \qquad \text{if } \mathsf{HeadTag}(\mathsf{exec}, \mathsf{f}(u_1, \ldots, u_n)) = \tau \text{ and } \tau \neq \bot \\ n_{\mathsf{f}(u_1,\ldots,u_n)}^{\epsilon,S} \text{ where } S = \mathsf{sameTagAs}(\mathsf{exec}, sid) \text{ otherwise} \end{cases}$

  for any $\mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$.

We extend our transformation on a trace in the expected way.

Definition 20 ($\overline{\mathsf{exec}}$)**.** Let Π be a protocol and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ an execution trace (not necessarily valid) of $\widetilde{\Pi}$. We define $\overline{\mathsf{exec}} = \overline{\mathsf{e}_1}^{\mathsf{exec},sid_1}; \ldots; \overline{\mathsf{e}_\ell}^{\mathsf{exec},sid_\ell}$, where

$$\overline{\mathsf{e}}^{\mathsf{exec},sid} = \begin{cases} \mathsf{P}(\overline{u_1}^{\mathsf{exec},sid}, \ldots, \overline{u_n}^{\mathsf{exec},sid}) & \text{if } \mathsf{e} = \mathsf{P}(u_1, \ldots, u_n) \\ \mathsf{snd}(\overline{u}^{\mathsf{exec},sid}) & \text{if } \mathsf{e} = \mathsf{snd}(u) \\ \mathsf{rcv}(\overline{u}^{\mathsf{exec},sid}) & \text{if } \mathsf{e} = \mathsf{rcv}(u) \end{cases}$$

With this transformation, we still get a trace associated to the protocol under study. Moreover, the resulting execution trace is well-formed. This is formally proved in Appendix A (Lemma 9 and Lemma 10).

Proposition 1. Let Π be a k-party protocol, and exec be an execution trace associated to $\widetilde{\Pi}$ (not necessarily a valid one). We have that $\overline{\mathsf{exec}}$ is a well-formed execution trace (not necessarily a valid one) associated to the protocol $\widetilde{\Pi}$.

6.3. **Validity.** Now, we show that the resulting execution trace, i.e. the one obtained by applying our transformation $\overline{\cdot}$, is still a valid one. In particular, we have to show that each term that occurs in a receive event is deducible from the initial knowledge of the attacker and the messages that have been sent so far. For this, we rely on the notion of simple proofs previously introduced in [17].

Definition 21 (simple proof)**.** Let $T_1 \subseteq T_2 \subseteq \cdots \subseteq T_n$. We say that a proof π of $T_i \vdash u$ is left-minimal if, whenever there is a proof of $T_j \vdash u$ for some $j < i$, then π is also a proof of $T_j \vdash u$. Then, we say that a proof π is simple if

(1) any subproof of π is left-minimal,
(2) a composition rule of the form $\dfrac{u_1 \quad u_2}{u}$ is never followed by a decomposition rule leading to u_1 or u_2, and
(3) any term of the form $\langle u_1, u_2 \rangle$ obtained by application of a decomposition rule or labelling a leaf is directly followed by a projection rule.

Example 11. Let $T_1 = \{n_1\}$ and $T_2 = \{n_1, \mathsf{encs}(\langle n_1, n_2 \rangle, k), k\}$. We have that $T_2 \vdash \langle n_1, n_2 \rangle$ with the proof tree π described below. However, π is not a simple proof of $T_2 \vdash \langle n_1, n_2 \rangle$. Indeed, the term $\langle n_1, n_2 \rangle$ has been obtained by an application of a decomposition rule. Thus, by Condition (3) of Definition 21 we have to decompose it. A simple proof of $T_2 \vdash \langle n_1, n_2 \rangle$ is the proof tree π' described below.

$$\pi = \left\{ \dfrac{\mathsf{encs}(\langle n_1, n_2 \rangle, k) \quad k}{\langle n_1, n_2 \rangle} \right. \qquad \pi' = \left\{ \dfrac{\dfrac{\mathsf{encs}(\langle n_1, n_2 \rangle, k) \quad k}{\langle n_1, n_2 \rangle} \quad n_2}{\langle n_1, n_2 \rangle} \right.$$

As it was done in [17] in a slightly different setting, we can show that it is always possible to consider such a proof tree, i.e. if there is a proof of $T_i \vdash u$, then there is a simple proof of it (w.r.t. a sequence $T_1 \subseteq T_2 \subseteq \cdots \subseteq T_n$). Given a simple proof π of $T_i \vdash u$, we can also show a locality lemma (by structural induction on π) allowing us to characterize the terms that occur in such a proof tree.

Lemma 3 (locality)**.** Let $T_1 \subseteq T_2 \subseteq \cdots \subseteq T_n$ be a set of terms and u be a term such that $T_i \vdash u$. Let π be a simple proof of $T_i \vdash u$. We have that π only involves terms in $\mathsf{St}(T_i \cup \{u\}) \cup \mathcal{K}_\epsilon \cup \mathcal{N}_\epsilon \cup \mathcal{A} \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\}$. Moreover, if π ends with an instance of a decomposition rule (or is reduced to a leaf), we have that π only involves terms in $\mathsf{St}(T_i) \cup \mathcal{K}_\epsilon \cup \mathcal{N}_\epsilon \cup \mathcal{A} \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\}$.

Now, relying on this notion of simple proof, we can show that deducibility is preserved by our transformation. This is the key lemma to ensure the validity of the resulting trace.

Lemma 4. Let Π be a k-party protocol and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a valid execution trace of $\widetilde{\Pi}$, w.r.t. some set T_0 of ground atoms. Let $i \in \{0, \ldots, \ell\}$ and t be a term such that $\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t$. We have that $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid}$ for any sid.

Proof. (sketch) Let $\mathsf{tr} = [\mathsf{ee}_1^{sid_1}; \ldots; \mathsf{ee}_\ell^{sid_\ell}]$ be the symbolic trace associated to exec and σ be the substitution such that $\mathrm{dom}(\sigma) = \mathit{vars}(\mathsf{tr})$ and $\mathsf{exec} = \mathsf{tr}\sigma$. Let $i \in \{0, \ldots, \ell\}$. Let π be a simple proof of $\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t$. We prove that $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid}$ by induction on (i, π). If $i = 0$ and π is a simple proof reduced to a leaf (possibly followed by some projection rules), then we have that $T_0 \vdash t$, and π is necessarily reduced to a leaf since T_0 only contains atomic terms. Let sid be a session identifier, we have that $\overline{t}^{\mathsf{exec},sid} \in \{t\} \cup \mathcal{N}_\epsilon$ since t is an atomic term. This allows us to conclude that $T_0 \vdash \overline{t}^{\mathsf{exec},sid}$. Now, we distinguish two cases depending on the last rule of π.

- *The proof π ends with an instance of a composition rule, i.e. $t = \mathsf{f}(t_1, \ldots, t_n)$ for some $\mathsf{f} \in \{\langle,\rangle, \mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$ and some terms $t_1, \ldots, t_n$.*

  According to Definition 19, we have that $\overline{t}^{\mathsf{exec},sid} \in \mathcal{N}^\epsilon \cup \{\mathsf{f}(\overline{t_1}^{\mathsf{exec},sid}, \ldots, \overline{t_n}^{\mathsf{exec},sid})\}$. If $\overline{t}^{\mathsf{exec},sid} \in \mathcal{N}_\epsilon$, we easily conclude that $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid}$. Otherwise, since π ends with a composition rule, we have that $\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t_1$, …, $\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t_n$. Moreover, the simple proofs witnessing these facts are strict subproofs of π that are also simple. Hence, we can apply our induction hypothesis and conclude that $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \mathsf{f}(\overline{t_1}^{\mathsf{exec},sid}, \ldots, \overline{t_n}^{\mathsf{exec},sid})$.

- *The proof ends with the application of a decomposition rule (but not a projection) possibly followed by several applications of the projection rules until the resulting term is not a pair.*

  We will here present the case of the symmetric decryption rule, but all the other decomposition rules (including the case of a proof reduced to a leaf) can be handled in a similar way. For some terms t_1 and t_2, the proof π is of the form

$$\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash \mathsf{encs}(t_1, t_2)} \quad \dfrac{\vdots}{\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t_2}}{\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t_1}}{\dfrac{\vdots}{\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t}}$$

  Let us first note that, by locality (Lemma 3) of π we know that $\mathsf{encs}(t_1, t_2) \in \mathsf{St}(\mathsf{K}(\mathsf{exec}_i)) \cup T_0 \cup \mathcal{K}_\epsilon \cup \mathcal{N}_\epsilon \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\}$, and by atomicity of T_0, $\mathcal{N}_\epsilon$, $\mathcal{K}_\epsilon$

and $\{\mathsf{pub}(a) \mid a \in \mathcal{A}\}$, we know that $\mathsf{encs}(t_1, t_2) \in \mathsf{St}(\mathsf{K}(\mathsf{exec}_i))$. (In case of a proof reduced to a leaf, and if there is no projection rule, we may have that $t \in T_0$. In such a case, as in the base case, we have that $T_0 \vdash \overline{t}^{\mathsf{exec},sid}$ and we easily conclude.) Hence, there exists $k \leq i$ such that $\mathsf{e}_k^{sid_k} = \mathsf{snd}(u)$ and $\mathsf{encs}(t_1, t_2) \in \mathsf{St}(u)$. Let k_0 be the smallest such k and u_0, u_0' be such that $\mathsf{e}_{k_0}^{sid_{k_0}} = \mathsf{snd}(u_0)$ and $\mathsf{ee}_{k_0}^{sid_{k_0}} = \mathsf{snd}(u_0')$. Hence, we have that $u_0 = u_0'\sigma$.

In order to prove the result, we first establish the following claim (proved in Appendix C).

Claim: We have that $\overline{\mathsf{encs}(t_1, t_2)}^{\mathsf{exec},sid_{k_0}} = \mathsf{encs}(\overline{t_1}^{\mathsf{exec},sid_{k_0}}, \overline{t_2}^{\mathsf{exec},sid_{k_0}})$.

Now, relying on this claim and applying the induction hypothesis, we have that:

- $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \mathsf{encs}(\overline{t_1}^{\mathsf{exec},sid_{k_0}}, \overline{t_2}^{\mathsf{exec},sid_{k_0}})$; and
- $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t_2}^{\mathsf{exec},sid_{k_0}}$.

This allows us to deduce that $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t_1}^{\mathsf{exec},sid_{k_0}}$. In order to establish that $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid}$, we need to distinguish two cases:

Case 1. $t \in \mathcal{A}$, $t = \mathsf{pub}(a)$ *or* $t = \mathsf{f}(a_1, \ldots, a_n)$ *for some* $\mathsf{f} \in \{\mathsf{shk}, \mathsf{priv}\}$. In such a case, we have that $\overline{t}^{\mathsf{exec},sid} = \overline{t}^{\mathsf{exec},sid_{k_0}} = t$. Hence, we have that $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid}$ by applying some projection rules on the proof of $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t_1}^{\mathsf{exec},sid_{k_0}}$.

Case 2. $t \in \mathcal{N}$ *or* $t = \mathsf{f}(t_1', \ldots, t_m')$ *for some* $\mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{h}, \mathsf{sign}\}$. First, if $\overline{t}^{\mathsf{exec},sid}$ can be obtained by application of some projection rules on the proof of $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t_1}^{\mathsf{exec},sid_{k_0}}$, then we easily conclude. Otherwise, it means that the term t is not abstracted in the same way in both cases. In such a case, we have that either $\overline{t}^{\mathsf{exec},sid} \in \mathcal{N}_\epsilon$ or $\overline{t}^{\mathsf{exec},sid_{k_0}} \in \mathcal{N}_\epsilon$. In the first case, we easily conclude. In the second case, i.e. $\overline{t}^{\mathsf{exec},sid_{k_0}} \in \mathcal{N}_\epsilon$ but $\overline{t}^{\mathsf{exec},sid} \notin \mathcal{N}_\epsilon$, we can show that t is a subterm of u_0 that either occurs as a component of u_0 or in the term $x\sigma$ for some $x \in vars(u_0')$. Actually, the first case is not possible since we have assumed that $\overline{t}^{\mathsf{exec},sid_{k_0}} \in \mathcal{N}_\epsilon$. Thus, only the second case remains. Thanks to the origination property, we know that t will occur in a previous receive event and we will be able to show that t was deducible using a smaller prefix of the trace allowing us to conclude by applying our induction hypothesis. □

Since our transformation preserves the deducibility relation, we can now prove the validity of the resulting trace by induction on the length of the original trace.

Proposition 2. Let Π be a k-party protocol and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a valid execution trace associated to $\widetilde{\Pi}$, w.r.t. some initial intruder knowledge T_0. We have that $\overline{\mathsf{exec}}$ is a well-formed and valid execution trace associated to $\widetilde{\Pi}$ w.r.t T_0.

Proof. First, according to Proposition 1, we know that $\overline{\mathsf{exec}}$ is an execution trace associated to $\widetilde{\Pi}$ which is well-formed. It remains to establish its validity w.r.t. T_0. We show by induction on i that for all $i \in \{1, \ldots, \ell\}$, $(\overline{\mathsf{exec}})_i$ is a valid execution trace. The base case, i.e. the empty trace $(\overline{\mathsf{exec}})_i = []$, is trivially valid. For the inductive step, we assume that $(\overline{\mathsf{exec}})_{\ell-1}$ is valid and we have to establish the validity of $\overline{\mathsf{exec}} = \overline{\mathsf{exec}}_\ell$. We distinguish 2 cases according to the nature of the last event in the trace.

Case $\mathsf{e}_\ell^{sid_\ell} = \mathsf{P}(t_1, \ldots, t_n)$ *or* $\mathsf{e}_\ell^{sid_\ell} = \mathsf{snd}(t)$. By induction hypothesis, we know that $(\overline{\mathsf{exec}})_{\ell-1}$ is a valid execution trace, and this is enough to conclude to the validity of $\overline{\mathsf{exec}}$.

Case $\mathsf{e}_\ell^{sid_\ell} = \mathsf{rcv}(t)$. By induction hypothesis, we know that $(\overline{\mathsf{exec}})_{\ell-1}$ is a valid execution trace. To conclude to the validity of $\overline{\mathsf{exec}}$, we only need to establish that $\mathsf{K}((\overline{\mathsf{exec}})_{\ell-1}) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid_\ell}$. Since we know that exec is a valid execution trace, we have that $\mathsf{K}(\mathsf{exec}_{\ell-1}) \cup T_0 \vdash t$. Applying Lemma 4, we conclude that $\mathsf{K}((\overline{\mathsf{exec}})_{\ell-1}) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid_\ell}$. This allows us to deduce that $\overline{\mathsf{exec}}$ is valid. □

6.4. **Satisfiability.** The goal of this section is to show that the trace $\overline{\mathsf{exec}}$ resulting of the application of our transformation will still satisfy the attack formula $\exists x_1. \ldots . \exists x_n . \phi$ under study. To show the validity of such a formula on the trace $\overline{\mathsf{exec}}$, we have to exhibit a substitution σ' for which $\langle \overline{\mathsf{exec}}, T_0 \rangle \models \phi\sigma'$. By hypothesis, we know that $\langle \mathsf{exec}, T_0 \rangle \models \phi\sigma$ for some σ. Thus, the idea is to consider the substitution $\sigma' = \{x_1 \mapsto \overline{x_1\sigma}^{\mathsf{exec},sid_1}, \ldots, x_n \mapsto \overline{x_n\sigma}^{\mathsf{exec},sid_n}\}$ where $sid_1, \ldots, sid_n$ correspond to the sessions from which the terms $x_1\sigma, \ldots, x_n\sigma$ come from.

Proposition 3. Let Π be a protocol, exec be an execution trace of $\widetilde{\Pi}$ w.r.t. some initial intruder knowledge T_0, and ϕ be an attack formula. We have that

$$\langle \mathsf{exec}, T_0 \rangle \models \phi \;\Rightarrow\; \langle \overline{\mathsf{exec}}, T_0 \rangle \models \phi.$$

The proof is done by structural induction on the formula and its details can be found in Appendix D. The technically difficult part is to formally link each variable existentially quantified in ϕ with the term it has been substituted with in order to satisfy the formula.

7. Second step: reducing the number of sessions

Now, our goal is to reduce the number of sessions that are involved in an execution trace witnessing the existence of an attack in order to match the bound announced in Theorem 1: the attack trace has to involved at most $\|\phi\|$ sessions of each role. The idea will be to identify a set of sessions S and to remove all the events that do not originate from a session in S according to the formal definition stated below.

Definition 22 (restriction of tr to S)**.** Let Π be a protocol, $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be an execution of Π, w.r.t. some set T_0 of ground atoms, and S be a set of session identifiers. The restriction of exec to S is defined as the trace $\mathsf{exec}|_S = [\mathsf{e}_{i_1}^{sid_{i_1}}; \ldots; \mathsf{e}_{i_h}^{sid_{i_h}}]$ satisfying the following: $i_1 < \ldots < i_h$ and for all $j \in \{1, \ldots, \ell\}$, there exists $k \in \{1, \ldots, h\}$ such that $j = i_k$ if and only if $sid_j \in S$.

Given a valid and well-formed execution exec and a set of sessions S, the goal of this section is to show that the restriction $\mathsf{exec}|_S$ is a valid and well-formed execution. Since messages coming from one session can be used to build a message for another session, to prove such a result, it is important to require some conditions on S. Basically, we will consider a set S that satisfies the following requirement:

for all sid_1 and sid_2 such that $\mathsf{sameTagAs}(\mathsf{exec}, sid_1) = \mathsf{sameTagAs}(\mathsf{exec}, sid_2)$,
we have that $sid_1 \in S$ if and only if $sid_2 \in S$.

This means that sessions using the same tag should have the same status w.r.t. the set S.

In the following of this section we will first show that *(i)* such a restricted execution is still a *valid* execution, and *(ii)* that the restriction preserves *satisfiability* of attack formulas.

7.1. **Validity of the restriction.** First, we show that in a well-formed and valid execution trace, terms that occur in sessions that are tagged differently do not share any name.

Lemma 5. Let Π be a k-party protocol, and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a well-formed valid execution of $\widetilde{\Pi}$ w.r.t. some set T_0 of ground atoms. Let $sess_1$ and $sess_2$ be two session identifiers. We have that:

$$\mathsf{sameTagAs}(\mathsf{exec}, sess_1) \neq \mathsf{sameTagAs}(\mathsf{exec}, sess_2)$$
$$\text{implies}$$
$$names(\mathsf{exec}, sess_1) \cap names(\mathsf{exec}, sess_2) = \emptyset$$

where $names(\mathsf{exec}, sess) = \{u \mid u \in names(\mathsf{e}_j^{sid_j}) \text{ for some } 1 \leq j \leq \ell \text{ such that } sid_j = sess\}$.

The goal of the next lemma is to show that deducibility is preserved when we consider the trace $\mathsf{exec}|_S$. Note that the previous lemma allows us to ensure that the terms we removed from the trace are "sufficiently disjoint" from the ones we keep. This is important to ensure that deducibility is preserved in the trace $\mathsf{exec}|_S$.

Lemma 6. Let Π be a k-party protocol, and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ a well-formed valid execution of $\widetilde{\Pi}$ w.r.t. some set T_0 of ground atoms, and such that $T_0 \cup \mathsf{K}(\mathsf{exec}) \not\vdash k$ for any $k \in lgKeys \setminus (\mathcal{K}_\epsilon \cup T_0)$ (exec does not reveal any long term keys). Let S be a set of sessions such that:

> for all session identifiers $sess_1$ and $sess_2$ such that $\mathsf{sameTagAs}(\mathsf{exec}, sess_1) = \mathsf{sameTagAs}(\mathsf{exec}, sess_2)$, we have that $sess_1 \in S$ if and only if $sess_2 \in S$.

For all term $t \in \mathsf{St}(\mathsf{exec}|_S)$ such that $T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t$, we have that $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t$.

Now, relying on Lemma 6, we are able to show that the trace $\mathsf{exec}|_S$ is valid.

Proposition 4. Let Π be a k-party protocol, and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ a well-formed valid execution of $\widetilde{\Pi}$ w.r.t. some set T_0 of ground atoms, and such that $T_0 \cup \mathsf{K}(\mathsf{exec}) \not\vdash k$ for any $k \in lgKeys \setminus (\mathcal{K}_\epsilon \cup T_0)$ (exec does not reveal any long term keys). Let S be a set of sessions such that:

> for all session identifiers $sess_1$ and $sess_2$ such that $\mathsf{sameTagAs}(\mathsf{exec}, sess_1) = \mathsf{sameTagAs}(\mathsf{exec}, sess_2)$, we have that $sess_1 \in S$ if and only if $sess_2 \in S$.

We have that $\mathsf{exec}|_S$ is also a well-formed and valid execution of $\widetilde{\Pi}$ w.r.t. T_0.

Proof. Let $1 \leq i_1 < \cdots < i_n \leq \ell$ such that $\mathsf{exec}|_S = [\mathsf{e}_{i_1}^{sid_{i_1}}; \ldots; \mathsf{e}_{i_n}^{sid_{i_n}}]$. We prove by induction on the length n of $\mathsf{exec}|_S$, that $\mathsf{exec}|_S$ is a valid execution of $\widetilde{\Pi}$, w.r.t T_0.

Base case: If $n = 0$ we have that $\mathsf{exec}|_S = []$, and thus $\mathsf{exec}|_S$ is a valid execution of $\widetilde{\Pi}$ w.r.t. T_0.

Inductive case: By induction hypothesis, we know that $[\mathsf{e}_{i_1}^{sid_{i_1}}; \ldots; \mathsf{e}_{i_{n-1}}^{sid_{i_{n-1}}}]$ is a valid execution of $\widetilde{\Pi}$ w.r.t. T_0. If $\mathsf{e}_n^{sid_n}$ is a send or a status event, then $\mathsf{exec}|_S = [\mathsf{e}_{i_1}^{sid_{i_1}}; \ldots; \mathsf{e}_{i_{n-1}}^{sid_{i_{n-1}}}; \mathsf{e}_{i_n}^{sid_{i_n}}]$ is a valid execution of $\widetilde{\Pi}$ w.r.t. T_0 (see Definition 7). On

the other hand, if $\mathsf{e}_n^{sid_n}$ is a receive event, i.e. $\mathsf{e}_n^{sid_n} = \mathsf{rcv}(t)$, we need to show that $T_0 \cup \mathsf{K}([\mathsf{e}_{i_1}^{sid_{i_1}}; \ldots; \mathsf{e}_{i_{n-1}}^{sid_{i_{n-1}}}]) \vdash t$ knowing that $T_0 \cup \mathsf{K}([\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_{(i_n)-1}^{sid_{(i_n)-1}}]) \vdash t$ which because e_{i_n} is a reception event implies that $T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t$. But then, according to Lemma 6 we know that $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t$. It suffices now to notice that by definition of $\mathsf{K}()$, because e_{i_n} is a reception event, we have that:

$$\mathsf{K}([\mathsf{e}_{i_1}^{sid_{i_1}}; \ldots; \mathsf{e}_{i_{n-1}}^{sid_{i_{n-1}}}]) = \mathsf{K}([\mathsf{e}_{i_1}^{sid_{i_1}}; \ldots; \mathsf{e}_{i_n}^{sid_{i_n}}]) = \mathsf{K}(\mathsf{exec}|_S).$$

This concludes the proof that $\mathsf{exec}|_S$ is a valid execution of $\widetilde{\Pi}$ w.r.t. the initial intruder knowledge T_0. Finally, it is obvious that $\mathsf{exec}|_S$ satisfies the 3 conditions of well-formedness (Definition 17), from the hypothesis that exec does. □

7.2. **Satisfiability of the formula.** The way the set S of sessions is chosen depends on the sessions that are needed to satisfy the attack formula under study. We therefore introduce the notion of *witness sessions* which for a given formula ϕ can be used to witness that ϕ holds.

Definition 23 (witness sessions, Ws)**.** Let Π be a protocol, ϕ a closed quantifier-free formula of $\mathcal{L}$, and T_0 be a set of ground atoms. Let $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a valid execution of Π (w.r.t. T_0) satisfying ϕ, i.e. $\langle \mathsf{exec}, T_0 \rangle \models \phi$. We define the set of sessions $\mathsf{Ws}(\mathsf{exec}, \phi)$ witnessing that $\langle \mathsf{exec}, T_0 \rangle \models \phi$ by structural induction on ϕ as follows:

- $\mathsf{Ws}(\mathsf{exec}, \neg\phi) = \mathsf{Ws}^-(\mathsf{exec}, \phi)$;
- $\mathsf{Ws}(\mathsf{exec}, \mathsf{true}) = \mathsf{Ws}(\mathsf{exec}, \mathsf{learn}(t)) = \mathsf{Ws}(\mathsf{exec}, \mathsf{C}(t)) = \emptyset$;
- $\mathsf{Ws}(\mathsf{Q}(t_1, \ldots, t_n)) = \{sid_\ell\}$;
- $\mathsf{Ws}(\mathsf{exec}, \Diamond\phi) = \mathsf{Ws}(\mathsf{exec}_i, \phi)$ where i is such that $\langle \mathsf{exec}_i, T_0 \rangle \models \phi$;
- $\mathsf{Ws}(\mathsf{exec}, \phi_1 \vee \phi_2) = \mathsf{Ws}(\mathsf{exec}, \phi_1)$ if $\langle \mathsf{exec}, T_0 \rangle \models \phi_1$ and $\mathsf{Ws}(\mathsf{exec}, \phi_2)$ otherwise;

where

- $\mathsf{Ws}^-(\mathsf{exec}, \neg\phi) = \mathsf{Ws}(\mathsf{exec}, \phi)$;
- $\mathsf{Ws}^-(\mathsf{exec}, \mathsf{true}) = \mathsf{Ws}^-(\mathsf{exec}, \mathsf{learn}(t)) = \mathsf{Ws}^-(\mathsf{exec}, \mathsf{C}(t)) = \mathsf{Ws}^-(\mathsf{exec}, \Diamond\phi) = \emptyset$;
- $\mathsf{Ws}^-(\mathsf{Q}(t_1, \ldots, t_n)) = \{sid_\ell\}$ when $\mathsf{length}(\mathsf{exec}) > 0$ and $\emptyset$ otherwise;
- $\mathsf{Ws}^-(\mathsf{exec}, \phi_1 \vee \phi_2) = \mathsf{Ws}^-(\mathsf{exec}, \phi_1) \cup \mathsf{Ws}^-(\mathsf{exec}, \phi_2)$.

Intuitively, we keep in the trace the sessions that are needed to satisfy the formula under study. Essentially, we have to keep those that are used to satisfy the status events occurring in the formula.

Lemma 7. Let Π be a k-party protocol, and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a valid and well-formed execution of $\widetilde{\Pi}$ w.r.t. some set T_0 of ground atoms such that $T_0 \cup \mathsf{K}(\mathsf{exec}) \not\vdash k$ for any $k \in \mathit{lgKeys} \setminus (\mathcal{K}_\epsilon \cup T_0)$. Let $\phi = \exists x_1 . \ldots . \exists x_n . \psi$ be an attack formula of $\mathcal{L}$, and σ be a ground substitution such that $\langle \mathsf{exec}, T_0 \rangle \models \psi\sigma$. Let S be a set of session identifiers such that:

(1) $\mathsf{Ws}(\mathsf{exec}, \psi\sigma) \subseteq S$, and
(2) $\forall sess_1, sess_2$ with $\mathsf{ExpectedTag}(\mathsf{exec}, sess_1) = \mathsf{ExpectedTag}(\mathsf{exec}, sess_2)$, we have that
$$sess_1 \in S \text{ if and only if } sess_2 \in S.$$

We have that $\mathsf{exec}|_S$ is an execution of $\widetilde{\Pi}$ that satisfies ϕ, i.e. $\langle \mathsf{exec}|_S, T_0 \rangle \models \phi$.

Proof. (sketch) The idea is to show that $\langle \mathsf{exec}|_S, T_0\rangle \models \psi\sigma$. However, this result is wrong in general since the substitution σ witnessing the fact that the attack formula ϕ is satisfiable can use some terms that only occur in events coming from sessions that are not in S. Thus, the first step of the proof consists in showing that we can consider a substitution σ that only involves subterms that occur in $\mathsf{St}(\mathsf{exec}|_S)$. For instance, consider the formula $\exists x.\mathsf{learn}(x)$. Since, the variable x does not occur in any status event, we cannot ensure that x will be bound to a term coming from a session in S. However, intuitively, we can replace such a term $x\sigma$ by a nonce in $\mathcal{N}_\epsilon$ still preserving the satisfiability of the attack formula. Now, we can assume w.l.o.g. that that for all $j \in \{1, \ldots, n\}$, $\sigma(x_j) \in \mathsf{St}(\mathsf{exec}, S) \cup \mathcal{A} \cup \mathit{lgKeys} \cup \mathcal{N}_\epsilon$. Then, we proceed by induction on the length of the execution trace and the size of the formula, and we show that $\langle \mathsf{exec}|_S, T_0\rangle \models \psi\sigma$. In other words, the attack formula is satisfiable and σ is a witness of this fact. □

Proposition 5. Let Π be a k-party protocol and T_0 be a finite set of ground atoms such that $\mathit{lgKeys}(\Pi) \cap \mathit{plaintext}(\Pi) \subseteq T_0 \cup \mathcal{K}_\epsilon$. Let exec be a valid and well-formed execution of $\widetilde{\Pi}$ w.r.t. T_0, and $\phi = \exists x_1. \ldots . \exists x_n.\psi$ be an attack formula such that $\langle \mathsf{exec}, T_0\rangle \models \psi\sigma$ for some ground substitution σ. We have that $\langle \mathsf{exec}|_S, T_0\rangle \models \phi$ where $S = \{\mathit{sid} \mid \exists \mathit{sid}' \in \mathsf{Ws}(\mathsf{exec}, \psi\sigma)$ and $\mathit{sid} \in \mathsf{sameTagAs}(\mathsf{exec}, \mathit{sid}')\}$.

Proof. Let exec be a valid and well-formed execution of $\widetilde{\Pi}$ w.r.t. T_0 such that $\langle \mathsf{exec}, T_0\rangle \models \phi$.

Claim: $T_0 \cup \mathsf{K}(\mathsf{exec}) \not\vdash k$ *for any* $k \in \mathit{lgKeys} \setminus (\mathcal{K}_\epsilon \cup T_0)$. Assume that there exists $k \in \mathit{lgKeys}$ such that $T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash k$. Using Lemma 1, we obtain that $k \in \mathit{plaintext}(\mathsf{exec}) \cup T_0 \cup \mathcal{K}_\epsilon$, and relying on Lemma 2, we conclude that $k \in \mathit{plaintext}(\mathsf{tr}) \cup T_0 \cup \mathcal{K}_\epsilon$ where tr is the symbolic trace underlying exec. Now, by construction of tr, if $k \in \mathit{plaintext}(\mathsf{tr})$, then there exists $k' \in \mathit{plaintext}(\Pi)$ such that $k = k'\sigma$ for some $\sigma : \mathcal{X} \to \mathcal{A}$. Hence, we have that $k' \in \mathit{lgKeys}(\Pi) \cup \mathit{plaintext}(\Pi)$. Thanks to our hypothesis, we conclude that $k' \in T_0 \cup \mathcal{K}_\epsilon$, and thus $k' = k \in T_0 \cup \mathcal{K}_\epsilon$, which concludes the proof of the claim.

By hypothesis, we have that $\langle \mathsf{exec}, T_0\rangle \models \psi\sigma$ for some ground substitution σ. Moreover, by hypothesis, we have that:

(1) $\mathsf{Ws}(\mathsf{exec}, \psi\sigma) \subseteq S$, and
(2) $\forall \mathit{sess}_1, \mathit{sess}_2$ with $\mathsf{ExpectedTag}(\mathsf{exec}, \mathit{sess}_1) = \mathsf{ExpectedTag}(\mathsf{exec}, \mathit{sess}_2)$, we have that
$$\mathit{sess}_1 \in S \text{ if and only if } \mathit{sess}_2 \in S.$$

Hence, we can apply Lemma 7 to conclude that $\langle \mathsf{exec}|_S, T_0\rangle \models \phi$. □

8. Main results

In this section, we put the pieces together and prove Theorem 1, the main result that was stated in Section 5.2. We also prove Corollary 1 which allows us to obtain slightly stronger results for particular security properties.

To prove our main result, we first need to bound the number of sessions that are needed to witness the satisfiability of the attack formula under study. This is the purpose of the following lemma that can be proved by induction on the structure of ϕ.

Lemma 8. Let Π be a protocol, ϕ a closed quantifier-free formula of $\mathcal{L}$, and T_0 be set of ground atoms. Let $\mathsf{exec} = [\mathsf{e}_1^{\mathit{sid}_1}; \ldots; \mathsf{e}_\ell^{\mathit{sid}_\ell}]$ be a valid execution of Π (w.r.t. T_0) satisfying ϕ, i.e. $\langle \mathsf{exec}, T_0\rangle \models \phi$. We have that $|\mathsf{Ws}(\mathsf{exec}, \phi)| \leq \|\phi\|$.

8.1. **Proof of Theorem 1.** Now, we prove our main theorem.

Theorem 1. Let Π be a k-party protocol, $\widetilde{\Pi}$ be its corresponding transformed protocol and T_0 be a set of ground atoms such that $\mathit{lgKeys}(\Pi) \cap \mathit{plaintext}(\Pi) \subseteq T_0 \cup \mathcal{K}_\epsilon$. Let ϕ be an attack formula such that $\widetilde{\Pi} \models \phi$ w.r.t. T_0. There exists a valid execution trace exec of $\widetilde{\Pi}$ such that:

$$\langle \mathsf{exec}, T_0 \rangle \models \phi \text{ and } \mathsf{exec} \text{ involves at most } \|\phi\| \text{ sessions of each role.}$$

Proof. Let exec be a valid execution of $\widetilde{\Pi}$ w.r.t. T_0 such that $\langle \mathsf{exec}, T_0 \rangle \models \phi$. By Proposition 2 we have that $\overline{\mathsf{exec}}$ is a valid well-formed execution of $\widetilde{\Pi}$ w.r.t. T_0, and according to Proposition 3, we have that $\langle \overline{\mathsf{exec}}, T_0 \rangle \models \phi$. By definition on an attack formula, we have that $\phi = \exists x_1 . \ldots . \exists x_n . \psi$ and we deduce that there exists σ such that $\langle \overline{\mathsf{exec}}, T_0 \rangle \models \psi\sigma$.

Let $S = \{sid \mid \exists sid' \in \mathsf{Ws}(\overline{\mathsf{exec}}, \psi\sigma) \text{ and } sid \in \mathsf{sameTagAs}(\overline{\mathsf{exec}}, sid')\}$. Now, by Proposition 4, we have that $\overline{\mathsf{exec}}|_S$ is also a well-formed and valid execution of $\widetilde{\Pi}$ w.r.t. T_0; and according to Proposition 5, we know that $\langle \overline{\mathsf{exec}}|_S, T_0 \rangle \models \phi$. Finally, Lemma 8 tells us that $|\mathsf{Ws}(\overline{\mathsf{exec}}, \psi\sigma)| \leq \|\phi\sigma\| = \|\phi\|$. But because by construction of $\widetilde{\Pi}$ (and hence of all of its symbolic traces), in every execution of $\widetilde{\Pi}$ all sessions of the same role are tagged differently (each session introduces its own nonce making them different), S must contain at most $\|\phi\|$ sessions of each role. This allows us to conclude that $\overline{\mathsf{exec}}|_S$ is an attack that involves at most $\|\phi\|$ sessions of each role. □

8.2. **Secrecy, aliveness and weak agreement.** For several classical security properties we are actually able to obtain a slightly stronger result and only consider one *honest* session of each role. As we will see below this is a direct corollary from the proof of the main theorem.

Corollary 1. Let Π be a k-party protocol, Π_{S} (respectively, Π_{A}, Π_{WA}) be the annotated protocol for modeling secrecy (respectively aliveness and weak agreement) as defined in Section 4.2.1, and $\widetilde{\Pi}_{\mathsf{S}}$ (respectively, $\widetilde{\Pi}_{\mathsf{A}}$, $\widetilde{\Pi}_{\mathsf{WA}}$) the corresponding transformed protocol. Let T_0 be a set of ground atoms such that $\mathit{lgKeys}(\Pi) \cap \mathit{plaintext}(\Pi) \subseteq T_0 \cup \mathcal{K}_\epsilon$ and $\overline{\phi_{\mathsf{S}}}$ (respectively $\overline{\phi_{\mathsf{A}}}$, $\overline{\phi_{\mathsf{WA}}}$) an attack formula against secrecy (respectively aliveness and weak agreement) as defined in Section 4.2.1. For $\mathsf{X} \in \{\mathsf{S}, \mathsf{A}, \mathsf{WA}\}$ we have that if $\widetilde{\Pi}_{\mathsf{X}} \models \overline{\phi_{\mathsf{X}}}$ w.r.t. T_0 then there exists a valid execution trace exec of $\widetilde{\Pi}_{\mathsf{X}}$ such that:

$$\langle \mathsf{exec}, T_0 \rangle \models \overline{\phi_{\mathsf{X}}} \text{ and } \mathsf{exec} \text{ involves at most one } \textit{honest} \text{ session of each role.}$$

Proof. We only detail the proof in the case of secrecy. The case of aliveness and weak agreement are treated similarly. Let $\overline{\phi_{\mathsf{S}}} = \exists x_1 . \ldots . \exists x_n . \exists y . \overline{\psi_{\mathsf{S}}}$. Following the proof of Theorem 1, we can show that $\langle \overline{\mathsf{exec}}, T_0 \rangle \models \overline{\psi_{\mathsf{S}}}\sigma$ for some substitution σ.

Let $S = \{sid \mid \exists sid' \in \mathsf{Ws}(\overline{\mathsf{exec}}, \overline{\psi_{\mathsf{S}}}\sigma) \text{ and } sid \in \mathsf{sameTagAs}(\overline{\mathsf{exec}}, sid')\}$. We have that $\mathsf{Ws}(\overline{\mathsf{exec}}, \overline{\psi_{\mathsf{S}}}\sigma) = 1$, and thus the set S contains at most *one* session of each role. To conclude, we have to show that S only contains *honest* sessions. By definition of Ws, we know that $\mathsf{Ws}(\overline{\mathsf{exec}}, \overline{\psi_{\mathsf{S}}}\sigma) = \{sid_0\}$ for some sid_0 such that the status event $\mathsf{Secret}(x_1\sigma, \ldots, x_k\sigma, y\sigma)$ is issued from the session sid_0 and we have that $\langle \overline{\mathsf{exec}}, T_0 \rangle \models \mathsf{NC}(x_1\sigma) \wedge \ldots \wedge \mathsf{NC}(x_k\sigma)$. Hence, we have that sid_0 is an honest session.

We have that $S = \{sid \mid \exists sid' \in \mathsf{Ws}(\overline{\mathsf{exec}}, \overline{\psi_{\mathsf{S}}}\sigma) \text{ and } sid \in \mathsf{sameTagAs}(\overline{\mathsf{exec}}, sid')\}$ which means that $S = \{sid \mid sid \in \mathsf{sameTagAs}(\overline{\mathsf{exec}}, sid_0)\}$. Since the names of the agents that are

involved in a session occur in the tag, we know that all the sessions in S are honest. This allows us to conclude. □

9. Conclusion

In this paper we present a transformation which guarantees that attacks on transformed protocols only require a number of sessions which is a function of the security property under study. We prove this result for a class of security properties that includes secrecy and several flavors of authentication. Our logic for specifying security properties does not allow one to express injective authentication properties (e.g. injective agreement, matching conversations, etc.) but we believe that both the logic and our reduction result could be extended to this setting.

A challenging topic for future research is to obtain more fine-grained characterizations of decidable classes of protocols for an unbounded number of sessions. The new insights gained by our work seem to be a good starting point to extract the conditions needed to reduce the security for an unbounded number of sessions to a finite number of sessions.

Acknowledgments

This work has been partially supported by the projects JCJC VIP ANR-11-JS02-006 and ERC grant agreement no 258865, project ProSecure.

References

[1] M. Arapinis. *Sécurité des protocoles cryptographiques : décidabilité et résultats de réduction.* Thèse de doctorat, Université Paris 12, Créteil, France, Nov. 2008.

[2] M. Arapinis, S. Delaune, and S. Kremer. From one session to many: Dynamic tags for security protocols. In I. Cervesato, H. Veith, and A. Voronkov, editors, *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08)*, volume 5330 of *Lecture Notes in Artificial Intelligence*, pages 128–142, Doha, Qatar, 2008. Springer.

[3] M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'07)*, volume 4855 of *LNCS*, pages 376–387. Springer, 2007.

[4] A. Armando et al. The Avispa tool for the automated validation of internet security protocols and applications. In *Proc. 17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.

[5] B. Barak, Y. Lindell, and T. Rabin. Protocol initialization for the framework of universal composability. Cryptology ePrint Archive, Report 2004/006, 2004.

[6] D. Beauquier and F. Gauche. How to guarantee secrecy for cryptographic protocols. *CoRR*, abs/cs/0703140, 2007.

[7] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *Proc. 30th Annual ACM Symposium on the Theory of Computing (STOC'98)*, pages 419–428. ACM Press, 1998.

[8] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 82–96, Cape Breton (Canada), 2001. IEEE Comp. Soc. Press.

[9] B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Proc. Foundations of Software Science and Computation Structures (FoSSaCS'03)*, volume 2620 of *LNCS*, pages 136–152. Springer, 2003.

[10] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS'01)*, pages 136–145, Las Vegas (Nevada, USA), 2001. IEEE Comp. Soc.

[11] C. Chevalier, S. Delaune, and S. Kremer. Transforming password protocols to compose. In S. Chakraborty and A. Kumar, editors, *Proceedings of the 31st Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'11)*, Leibniz International Proceedings in Informatics, pages 204–216. Leibniz-Zentrum für Informatik, 2011.

[12] Ş. Ciobâcă and V. Cortier. Protocol composition for arbitrary primitives. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 322–336. IEEE Computer Society Press, July 2010.

[13] J. Clark and J. Jacob. A survey of authentication protocol literature. `http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps`, 1997.

[14] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. *Science of Computer gramming*, 50(1-3):51–71, March 2004.

[15] H. Comon-Lundh, V. Cortier, and E. Zălinescu. Deciding security properties for cryptographic protocols. application to key cycles. *ACM Trans. Comput. Logic*, 11(2):1–42, 2010.

[16] R. Corin. *Analysis Models for Security Protocols*. PhD thesis, University of Twente, 2006.

[17] V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, feb 2009.

[18] V. Cortier, B. Warinschi, and E. Zălinescu. Synthesizing secure protocols. In *Proc. 12th European Symposium On Research In Computer Security (ESORICS'07)*, volume 4734 of *LNCS*, pages 406–421. Springer, 2007.

[19] D. Dolev, S. Even, and R. M. Karp. On the security of ping-pong protocols. In *Proc. Advances in Cryptology (CRYPTO'82)*, pages 177–186, 1982.

[20] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proc. of the 22nd Symposium on Foundations of Computer Science (FOCS'81)*, pages 350–357. IEEE Comp. Soc. Press, 1981.

[21] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on Formal Methods and Security Protocols*, 1999.

[22] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proc. 13th Computer Security Foundations Workshop (CSFW'01)*, pages 255–268. IEEE Comp. Soc. Press, 2000.

[23] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. In *Proc. 23rd Annual International Cryptology Conference (CRYPTO'03)*, volume 2729 of *LNCS*, pages 110–125. Springer, 2003.

[24] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166, Berlin (Germany), 1996. Springer.

[25] G. Lowe. A hierarchy of authentication specifications. In *CSFW '97: Proceedings of the 10th IEEE workshop on Computer Security Foundations*, page 31, Washington, DC, USA, 1997. IEEE Computer Society.

[26] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(1), 1999.

[27] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communication of the ACM*, 21(12):993–999, 1978.

[28] R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable for unbounded nonces as well. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS'03)*, volume 2914 of *LNCS*, pages 363–374. Springer, 2003.

[29] R. Ramanujam and S. P. Suresh. Decidability of context-explicit security protocols. *Journal of Computer Security*, 13(1):135–165, 2005.

[30] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions and composed keys is NP-complete. *Theoretical Computer Science*, 299(1-3):451–475, 2003.

Appendix A. Proofs of Section 6.2

In this section, we show that our transformation maps an execution trace exec to a well-formed execution trace $\overline{\mathsf{exec}}$. The resulting execution trace $\overline{\mathsf{exec}}$ is still a trace associated to the protocol $\widetilde{\Pi}$ under study.

Lemma 9. Let Π be a k-party protocol, and exec be an execution trace associated to $\widetilde{\Pi}$ (not necessarily a valid one). We have that $\overline{\mathsf{exec}}$ is an execution trace (not necessarily a valid one) associated to the protocol $\widetilde{\Pi}$

Proof. (sketch) Let $\mathsf{tr} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be the symbolic trace of $\widetilde{\Pi}$, and σ be the ground substitution such that $\mathrm{dom}(\sigma) = vars(\mathsf{tr})$ and $\mathsf{exec} = \mathsf{tr}\sigma$. Let $\overline{\sigma}$ be such that:

- $\mathrm{dom}(\overline{\sigma}) = \mathrm{dom}(\sigma)$, and
- $\overline{\sigma}(x) = \overline{x\sigma}^{\mathsf{exec},sid}$ where $x \in vars(\mathsf{tr}, sid)$.

Clearly, we have that $\overline{\sigma}$ is a ground substitution. It remains to establish that $\overline{\mathsf{exec}} = \mathsf{tr}\overline{\sigma}$ so that the execution exec will rely on the same scenario than exec.

By definition $\overline{\mathsf{exec}} = [\overline{\mathsf{e}_1^{sid_1}\sigma}^{\mathsf{exec},sid_1}; \ldots; \overline{\mathsf{e}_\ell^{sid_\ell}\sigma}^{\mathsf{exec},sid_\ell}]$. Let $i \in \{1, \ldots, \ell\}$, then we have that $\mathsf{e}_i^{sid_i} = \mathsf{rcv}(u)$ (or $\mathsf{e}_i^{sid_i} = \mathsf{snd}(u)$, or $\mathsf{e}_i^{sid_i} = \mathsf{Q}(u_1, \ldots, u_n)$). Since the three cases can be handled in a similar way, we consider here the case where $\mathsf{e}_i^{sid_i} = \mathsf{rcv}(u)$. By definition, we have that $\overline{\mathsf{e}_i^{sid_i}\sigma}^{\mathsf{exec},sid_i} = \mathsf{rcv}(\overline{u\sigma}^{\mathsf{exec},sid_i})$, and we prove by structural induction on $u' \in \mathsf{St}(u)$ that $\overline{u'\sigma}^{\mathsf{exec},sid_i} = u'\overline{\sigma}$. Finally, from this we conclude that $\overline{u\sigma}^{\mathsf{exec},sid_i} = u\overline{\sigma}$, and thus that $\overline{\mathsf{e}_i^{sid_i}\sigma}^{\mathsf{exec},sid_i} = \mathsf{rcv}(\overline{u\sigma}^{\mathsf{exec},sid_i}) = \mathsf{rcv}(u\overline{\sigma}) = \mathsf{e}_i^{sid_i}\overline{\sigma}$. By definition, this brings us to $\overline{\mathsf{exec}} = \mathsf{tr}\overline{\sigma}$. □

Lemma 10. Let Π be a k-party protocol, and exec be an execution trace associated to $\widetilde{\Pi}$ (not necessarily a valid one). We have that $\overline{\mathsf{exec}}$ is well-formed.

Proof. (sketch) Let $\mathsf{exec} = \mathsf{e}_1^{sid_1}, \ldots, \mathsf{e}_\ell^{sid_\ell}$. Let $i \in \{1, \ldots, \ell\}$, we show that:

(1) $\overline{\mathsf{e}_i^{sid_i}}^{\mathsf{exec},sid_i}$ is k-tagged;
(2) $\mathsf{Tags}(\overline{\mathsf{e}_i^{sid_i}}^{\mathsf{exec},sid_i}) \subseteq \{\mathsf{ExpectedTag}(\overline{\mathsf{exec}}, sid_i)\}$;
(3) $names(\overline{\mathsf{e}_i^{sid_i}}^{\mathsf{exec},sid_i}) \subseteq \{n_t^{\epsilon,S} \mid t \in T\} \cup \{n_y^{sid} \mid sid \in S \text{ and } y \in \mathcal{Y}\}$ where $S = \mathsf{sameTagAs}(\mathsf{exec}, sid_i)$.

Let $\mathsf{e}_i^{sid_i} = \mathsf{rcv}(u)$ for some term u. The cases where $\mathsf{e}_i^{sid_i} = \mathsf{snd}(u)$ or $\mathsf{e}_i^{sid_i} = \mathsf{Q}(u_1, \ldots, u_n)$ can be done in a similar way. We have that $\overline{\mathsf{e}_i^{sid_i}}^{\mathsf{exec},sid_i} = \mathsf{rcv}(\overline{u}^{\mathsf{exec},sid_i})$ and we prove by structural induction on $u' \in \mathsf{St}(u)$ that:

(1) $\overline{u'}^{\mathsf{exec},sid_i}$ is k-tagged;
(2) $\mathsf{Tags}(\overline{u'}^{\mathsf{exec},sid_i}) \subseteq \{\mathsf{ExpectedTag}(\overline{\mathsf{exec}}, sid_i)\}$;
(3) $names(\overline{u'}^{\mathsf{exec},sid_i}) \subseteq \{n_t^{\epsilon,S} \mid t \in T\} \cup \{n_y^{sid} \mid sid \in S \text{ and } y \in \mathcal{Y}\}$ where $S = \mathsf{sameTagAs}(\mathsf{exec}, sid_i)$.

And from this we derive that $\overline{u}^{\mathsf{exec},sid_i}$ satisfies the three conditions of well-formedness, and thus so is $\overline{\mathsf{e}_i^{sid_i}}^{\mathsf{exec},sid_i}$ for all $i \in \{1,\ldots,\ell\}$, which in turn implies by definition that $\overline{\mathsf{exec}}$ satisfies the three conditions of well-formedness and is thus well-formed. □

Appendix B. Technical proofs about alien subterms

We introduce the notion of alien subterms and we show that they satisfy some good properties. Later on, we will see that those alien subterms correspond to the subterms that are abstracted by our transformation $\overline{\cdot}$ and we will use the properties established on them to prove the validity of the trace obtained after transformation.

Definition 24 ($\mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,t)$)**.** Let Π be a k-party protocol and $\mathsf{exec} = [\mathsf{e}_1^{sid_1};\ldots;\mathsf{e}_\ell^{sid_\ell}]$ be an execution trace (not necessarily valid) of $\widetilde{\Pi}$. We define the alien subterms of a term t w.r.t. the execution exec and the active tag τ, denoted $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,t)$, as follows:

- $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,n) = \{n\}$ if $n \in \mathcal{N}_\epsilon$
- $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,n_y^{sid}) = \begin{cases} \emptyset & \text{if } \mathsf{ExpectedTag}(\mathsf{exec},sid) = \tau \text{ and } \tau \neq \bot \\ \{n_y^{sid}\} & \text{otherwise} \end{cases}$
- $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,a) = \emptyset$ if a is an agent name
- $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,\mathsf{f}(a_1,\ldots,a_n)) = \emptyset$ if $\mathsf{f} \in \{\mathsf{shk},\mathsf{pub},\mathsf{priv}\}$
- $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,\langle u,v\rangle) = \mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,u) \cup \mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,v)$
- $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,\mathsf{f}(u_1,\ldots,u_n)) = \begin{cases} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,u_1) \cup \cdots \cup \mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau,u_n) \\ \text{if } \mathsf{HeadTag}(\mathsf{exec},\mathsf{f}(u_1,\ldots,u_n)) = \tau \text{ and } \tau \neq \bot \\ \{\mathsf{f}(u_1,\ldots,u_n)\} \cup \bigcup_{i\in\{1,\ldots,n\}} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\tau',u_i) \\ \text{otherwise where } \tau' = \mathsf{HeadTag}(\mathsf{exec},\mathsf{f}(u_1,\ldots,u_n)) \end{cases}$

  if $\mathsf{f} \in \{\mathsf{encs},\mathsf{enca},\mathsf{sign},\mathsf{h}\}$.

We define $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec},t) = \mathsf{St}_{\mathsf{alien}}(\mathsf{exec},\bot,t)$, and extend this notion to sets of terms in the obvious way, i.e. $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec},T) = \bigcup_{t\in T} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec},t)$.

Definition 25 ($vars(\mathsf{exec},sid)$, $\mathsf{St}(\mathsf{exec},sid)$, $names(\mathsf{exec},sid)$)**.** Let Π be a k-party protocol and exec be an execution trace of $\widetilde{\Pi}$. Let sid be a session identifier, and $[\mathsf{e}_1^{sid};\ldots;\mathsf{e}_h^{sid}] \stackrel{\mathsf{def}}{=} \mathsf{exec}|_{\{sid\}}$. We define the variables, subterms, and names in exec of a session sid as follows:

$$\begin{aligned} vars(\mathsf{exec},sid) &= \{x \mid x \in vars(\mathsf{e}_j^{sid}) \text{ for some } j \in \{1,\ldots,h\}\} \\ \mathsf{St}(\mathsf{exec},sid) &= \{u \mid u \in \mathsf{St}(\mathsf{e}_j^{sid}) \text{ for some } j \in \{1,\ldots,h\}\} \\ names(\mathsf{exec},sid) &= \{u \mid u \in names(\mathsf{e}_j^{sid}) \text{ for some } j \in \{1,\ldots,h\}\}. \end{aligned}$$

Since we do not tag the pairing function symbol, this function symbol has a special status. We denote by $\mathsf{comp}(t)$ the components of a term t. This notion is formally defined as follows:

Definition 26 ($\mathsf{comp}(t)$)**.** Let t be a term, the set of components of t is:

$$\mathsf{comp}(t) = \begin{cases} \mathsf{comp}(u) \cup \mathsf{comp}(v) & \text{if } t = \langle u,v\rangle \\ \{t\} & \text{otherwise.} \end{cases}$$

Lemma 11. Let Π be a k-party protocol, exec be an execution trace of $\widetilde{\Pi}$, and t be a term. For all k-tags τ, we have that:

(1) $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, t) = \bigcup_{t' \in \mathsf{comp}(t)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, t')$;
(2) $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, t) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$;
(3) $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t) \subseteq \mathsf{comp}(t) \cup \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, t)$.

Proof. We prove each statement separately by induction on the depth of t. □

Lemma 12. Let Π be a k-party protocol, exec be an execution trace of $\widetilde{\Pi}$, and u be a term. For any $v \in \mathsf{St}(u)$, we have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, v) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, u) \cup \mathsf{comp}(v)$.

Proof. We first need to establish the following result:

$$\forall \tau' \; \exists \; \tau \; \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, v) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', u).$$

If $v = u$ then we can choose $\tau = \tau'$ to prove what we want. Otherwise, we have that $v \neq u$, and we prove the result by induction on the depth of u, and for this we distinguish three cases:

Case $u = \mathsf{f}(u_1, \ldots, u_n)$ *for some* $\mathsf{f} \in \{\mathsf{pub}, \mathsf{priv}, \mathsf{shk}\}$. In such a case, we have that $\mathsf{St}_{\mathsf{alien}}(v) = \emptyset$ for any $v \in \mathsf{St}(u)$. This allows us to easily conclude.

Case $u = \langle u_1, u_2 \rangle$. In that case $v \in \mathsf{St}(u_1)$ or $v \in \mathsf{St}(u_2)$. Suppose $v \in \mathsf{St}(u_1)$ and let τ' be a tag. By induction hypothesis, we have that there exists τ such that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, v) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', u_1)$. By Definition 24, we have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', u_1) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', u)$. Hence, we easily conclude. The case where $v \in \mathsf{St}(u_2)$ can be handled in a similar way.

Case $u = \mathsf{f}(u_1, \ldots, u_n)$ *for some* $\mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$. In that case, we have that $v \in \mathsf{St}(u_{i_0})$ for some $i_0 \in \{1, \ldots, n\}$. Let τ' be a k-tag. According to Definition 24, we have that $\bigcup_{i \in \{1,\ldots,n\}} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau'', u_i) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', u)$ where $\tau'' = \mathsf{HeadTag}(\mathsf{exec}, u)$. Moreover, by induction hypothesis, we know that there exists τ such that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, v) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau'', u_{i_0})$. Hence, we deduce that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, v) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', u)$.

This allows us to conclude that $\forall \tau' \; \exists \tau, \; \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, v) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', u)$. We have shown that $\forall \tau' \; \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', u) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, u)$ (see Lemma 11 - Item 2). Hence, we can infer that there exists τ such that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, v) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, u)$. Hence, we have that:

$$\begin{array}{rcll} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, v) & \subseteq & \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, v) \cup \mathsf{comp}(v) & \text{(Lemma 11 - Item 3)} \\ & \subseteq & \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, u) \cup \mathsf{comp}(v) & \end{array}$$

□

Lemma 13. Let Π be a k-party protocol and exec be an execution trace of $\widetilde{\Pi}$. Let T be a set of terms such that $T \vdash v$ for any $v \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, T)$, and t be a term such that $T \vdash t$. We have $T \vdash u$ for any $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$.

Proof. Let $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$. We prove that $T \vdash u$ by induction on π, a prooftree witnessing the fact that $T \vdash t$. If π is reduced to a leaf then we have that $t \in T \cup \mathcal{A} \cup \mathcal{K}_\epsilon \cup \mathcal{N}_\epsilon \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\}$. Actually, if $t \in \mathcal{A} \cup \mathcal{K}_\epsilon \cup \mathcal{N}_\epsilon \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\}$, then $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t) = \emptyset$, leading to a contradiction. Hence, we have that $t \in T$, and thus $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, T)$. We can thus conclude by hypothesis that $T \vdash u$.

Otherwise, we proceed by case analysis on the last rule used in the proof π.

Case 1: the last rule is a composition rule. Then $t = \mathsf{f}(t_1, \ldots, t_n)$ for some terms $t_1, \ldots, t_n$ and some $\mathsf{f} \in \{\langle,\rangle, \mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$. Let $\pi_1, \ldots, \pi_n$ be the direct subproofs of π. We have that π_i is a proof of $T \vdash t_i$ for $i \in \{1, \ldots, n\}$. According to Definition 24 of alien subterms, $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t) \subseteq \{t\} \cup \bigcup_{i\in\{1,\ldots,n\}} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, t_i)$ for some τ, and by Lemma 11 (Item 2) we can thus infer that

$$\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t) \subseteq \{t\} \cup \bigcup_{i\in\{1,\ldots,n\}} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t_i).$$

If $u = t$, then by hypothesis we know that $T \vdash u$. On the other hand , if $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t_i)$ for some $i \in \{1, \ldots, n\}$, then we conclude by applying our induction hypothesis on π_i. In both cases, we have that $T \vdash u$.

Case 2: the last rule is a projection rule. Then $t = t_{i_0}$ for some terms t_1, t_2, and some $i_0 \in \{1, 2\}$. Let π' be the direct subproof of π. We have that π is a proof of $T \vdash \langle t_1, t_2\rangle$. According to Definition 24 of alien subterms, $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \langle t_1, t_2\rangle)$, i.e. $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \langle t_1, t_2\rangle)$. We can thus conclude by applying our induction hypothesis on π' that $T \vdash u$.

Case 3: the last rule is another decomposition rule. In such a case, there exists t' such that one of the direct subproofs of π is labeled with $\mathsf{f}(t, t')$. Let π' be such a proof. Thanks to Lemma 12 we know that either $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \mathsf{f}(t, t'))$ or $u \in \mathsf{comp}(t)$. In the first case, we can conclude by applying our induction hypothesis on π' that $T \vdash u$. In the second case, we know that by application of the projection rules one can derive u from t, hence $T \vdash u$. □

Lemma 14. Let Π be a k-party protocol and exec be an execution trace of $\widetilde{\Pi}$ associated to the symbolic trace $\mathsf{tr} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$. Let σ be the substitution such that $\mathrm{dom}(\sigma) = \mathit{vars}(\mathsf{tr})$ and $\mathsf{exec} = \mathsf{tr}\sigma$.

$$\forall t \in \mathsf{St}(\mathsf{tr}, \mathit{sid}) \quad \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) \subseteq \mathsf{comp}(t\sigma) \cup \bigcup_{x\in \mathit{vars}(t)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$$

where $\tau = \mathsf{ExpectedTag}(\mathsf{exec}, \mathit{sid})$.

Proof. Let $t \in \mathsf{St}(\mathsf{tr}, \mathit{sid})$ and $\tau = \mathsf{ExpectedTag}(\mathsf{exec}, \mathit{sid})$. We show by structural induction on t that

$$\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) \subseteq \mathsf{comp}(t\sigma) \cup \bigcup_{x\in \mathit{vars}(t)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$$

We distinguish several cases.

Case $t \in \mathcal{Y}$. In such a case, we can easily conclude thanks to Lemma 11 (Item 3). Indeed, we have that:

$$\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) \subseteq \mathsf{comp}(t\sigma) \cup \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, t\sigma) = \mathsf{comp}(t\sigma) \cup \bigcup_{x\in \mathit{vars}(t)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$$

Case $t \in \mathcal{N}$. Then $t\sigma = t$, $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) = \{t\sigma\}$, and $\mathsf{comp}(t\sigma) = \{t\sigma\}$. Thus, we have that:

$$\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) = \{t\sigma\} \subseteq \mathsf{comp}(t\sigma) \cup \bigcup_{x\in \mathit{vars}(t)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$$

Case $t \in \mathcal{A}$ *or* $t = \mathsf{f}(a_1, \ldots, a_n)$ *for some* $\mathsf{f} \in \{\mathsf{shk}, \mathsf{pub}, \mathsf{priv}\}$. In such a case, $vars(t) = \emptyset$ and thus $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) = \emptyset$. This allows us to conclude.

Case $t = \langle t_1, t_2 \rangle$.In such a case, we have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) = \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t_1\sigma) \cup \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t_2\sigma)$. Applying our induction hypothesis, we deduce that

$$\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t_i\sigma) \subseteq \mathsf{comp}(t_i\sigma) \cup \bigcup_{x \in vars(t_i)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma) \text{ for } i \in \{1, 2\}.$$

Hence, we conclude that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) \subseteq \mathsf{comp}(t\sigma) \cup \bigcup_{x \in vars(t)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$

Case $t = \mathsf{f}(t_1, \ldots, t_n)$ *for some* $\mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$. Let $\tau' = \mathsf{HeadTag}(\mathsf{exec}, t\sigma)$. We have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) = \{t\sigma\} \cup \bigcup_{i \in \{1,\ldots,n\}} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', t_i\sigma)$. By construction of tr, for all subterms $u \in \mathsf{CryptSt}(\mathsf{tr}, sid)$, $\mathsf{HeadTag}(\mathsf{exec}, u\sigma) = \tau$, thus $\tau' = \tau$. Thanks to Lemma 11 (Item 2), we have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) \subseteq \{t\sigma\} \cup \bigcup_{i \in \{1,\ldots,n\}} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t_i\sigma)$. We have that $\mathsf{comp}(t\sigma) = \{t\sigma\}$ and thanks to our induction hypothesis we have for each $i \in \{1, \ldots, n\}$ the following inclusion

$$\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t_i\sigma) \subseteq \mathsf{comp}(t_i\sigma) \cup \bigcup_{x \in vars(t_i)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$$

Thus, $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) \subseteq \mathsf{comp}(t\sigma) \cup \bigcup_{i \in \{1,\ldots,n\}} \mathsf{comp}(t_i\sigma) \cup \bigcup_{x \in vars(t)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$.

Now, in order to conclude, it remains to show that for all $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma)$, if $u \in \bigcup_{i \in \{1,\ldots,n\}} \mathsf{comp}(t_i\sigma)$ then there exists $x \in vars(t)$ such that $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$. First, we notice the following:

$$\begin{array}{rcll}
\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) & = & \{t\sigma\} \cup \bigcup_{i \in \{1,\ldots,n\}} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', t_i\sigma) & \text{(Definition 24)} \\
& = & \{t\sigma\} \cup \bigcup_{i \in \{1,\ldots,n\}} \bigcup_{w \in \mathsf{comp}(t_i\sigma)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', w) & \text{(Lemma 11)} \\
& = & \{t\sigma\} \cup \bigcup_{i \in \{1,\ldots,n\}} \bigcup_{v \in \mathsf{comp}(t_i)} \bigcup_{w \in \mathsf{comp}(v\sigma)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', w) & \text{(Definition 26)} \\
& = & \{t\sigma\} \cup \bigcup_{i \in \{1,\ldots,n\}} \bigcup_{v \in \mathsf{comp}(t_i)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', v\sigma) & \text{(Lemma 11)}
\end{array}$$

Let $i \in \{1, \ldots, n\}$ be such that $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma)$ and $u \in \mathsf{comp}(t_i\sigma)$. In that case, according to the equation stated above, there exists $j \in \{1, \ldots, n\}$ such that $v \in \mathsf{comp}(t_j)$ and $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', v\sigma)$. We now proceed by case analysis on v:

- *Case* $v \in \mathcal{A}$ *or* $v = \mathsf{f}(a_1, \ldots, a_n)$ *for some* $f \in \{\mathsf{pub}, \mathsf{priv}, \mathsf{shk}\}$. In such a case, we have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', v\sigma) = \emptyset$. Thus, this case in not possible.
- *Case* $v \in \mathcal{N}$. In such a case, we have that $u = v$ and by construction of tr we have that $v = n_y^{sid}$ for some variable y. Since, $\tau = \tau'$, we have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', v\sigma) = \emptyset$. Thus, this case is not possible.
- *Case* $v = \mathsf{g}(v_1, \ldots, v_m)$ *for some* $\mathsf{g} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$. In such a case, we have that $u = v\sigma$ and by construction of tr we know that $\mathsf{HeadTag}(\mathsf{exec}, v\sigma) = \mathsf{HeadTag}(\mathsf{exec}, t\sigma) = \tau (= \tau')$. Hence, we deduce that $v\sigma \notin \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', v\sigma)$, and thus $u \notin \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau', v\sigma)$ leading again to a contradiction.
- *Case* v *is a variable.* In such a case, we have that $v \in vars(t_j) \subseteq vars(t)$. Hence, we have the expected conclusion.

Altogether, this allows us to conclude that

$$\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t\sigma) \;\subseteq\; \mathsf{comp}(t\sigma) \;\cup \bigcup_{x \in vars(t)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma). \qquad \square$$

Lemma 15. Let Π be a k-party protocol and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be an execution trace of $\widetilde{\Pi}$, w.r.t. some set T_0 of ground atoms, associated to the symbolic trace $\mathsf{tr} = [\mathsf{ee}_1^{sid_1}; \ldots; \mathsf{ee}_\ell^{sid_\ell}]$. Let σ be the substitution such that $\mathrm{dom}(\sigma) = vars(\mathsf{tr})$ and $\mathsf{exec} = \mathsf{tr}\sigma$. Let sid be a session identifier, x be a variable in $vars(\mathsf{tr}, sid)$, $\tau = \mathsf{ExpectedTag}(\mathsf{exec}, sid)$, and $u \in \mathsf{St}(\mathsf{tr})$ such that $x \in vars(u)$. We have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, u\sigma)$.

Proof. Let u be a subterm of tr such that $x \in vars(u)$. We prove the result by structural induction on u. First, note that by construction of tr, we have that $vars(\mathsf{tr}, sid') \cap vars(\mathsf{tr}, sid'') = \emptyset$ when $sid' \neq sid''$. Hence, for all $i \in \{1, \ldots, \ell\}$ if $x \in vars(\mathsf{ee}_i^{sid_i})$, then $sid_i = sid$; and thus, for all $i \in \{1, \ldots, \ell\}$ such that $u \in \mathsf{St}(\mathsf{ee}_i^{sid_i})$, we know that $sid_i = sid$. Now, since $x \in vars(u)$, we know that u is not ground, and we only need to consider the three following cases:

Case $u \in \mathcal{Y}$. In this case $u = x$, and the result trivially holds.

Case $u = \langle u_1, u_2 \rangle$ *for some terms* u_1 *and* u_2. In that case, $x \in vars(u_1)$ or $x \in vars(u_2)$. Assume that $x \in vars(u_1)$. The other case can be handled in a similar way. By induction hypothesis, we know that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, u_1\sigma)$ and we have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, u_i\sigma) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, u\sigma)$. Combining these two we easily conclude.

Case $u = \mathsf{f}(u_1, \ldots, u_n)$ *for some* $\mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$ *and some terms* $u_1, \ldots, u_n$. In that case $x \in vars(u_i)$ for some $i \in \{1, \ldots, n\}$. Let $j \in \{1, \ldots, n\}$ such that $x \in vars(u_j)$. Now, by construction of tr, we know that $\mathsf{HeadTag}(\mathsf{exec}, u\sigma) = \mathsf{ExpectedTag}(\mathsf{exec}, sid)$, hence we have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, u_j\sigma) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, u\sigma)$. Applying our induction hypothesis on u_j, we deduce that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, u_j\sigma)$. This allows us to conclude. $\square$

Now, we can show that the alien subterms that occur in a valid trace are deducible.

Lemma 16. Let Π be a k-party protocol and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be an execution trace of $\widetilde{\Pi}$ that is valid w.r.t. some set T_0 of ground atoms. Let $i \in \{0, \ldots, \ell\}$. We have that $\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash u$ for any $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \mathsf{K}(\mathsf{exec}_i) \cup T_0)$.

Proof. Let $\mathsf{tr} = [\mathsf{ee}_1^{sid_1}; \ldots; \mathsf{ee}_\ell^{sid_\ell}]$ be the symbolic trace associated to exec. Let σ be the substitution such that $\mathrm{dom}(\sigma) = vars(\mathsf{tr})$ and $\mathsf{exec} = \mathsf{tr}\sigma$. We prove the result by induction on i. The base case, where $i = 0$ is obvious since $\mathsf{K}(\mathsf{exec}_i) = \emptyset$ and $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, T_0) \subseteq T_0$. Now, to deal with the inductive case, we distinguish three cases depending on the nature of the last event in exec_i.

Case $\mathsf{e}_i^{sid_i} = \mathsf{P}(t_1, \ldots, t_n)$. Then, we have that $\mathsf{K}(\mathsf{exec}_i) = \mathsf{K}(\mathsf{exec}_{i-1})$ and thus that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \mathsf{K}(\mathsf{exec}_i) \cup T_0) = \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \mathsf{K}(\mathsf{exec}_{i-1}) \cup T_0)$. Thanks to our induction hypothesis, we know that $\mathsf{K}(\mathsf{exec}_{i-1}) \cup T_0 \vdash \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \mathsf{K}(\mathsf{exec}_{i-1}) \cup T_0))$, thus we easily conclude.

Case $\mathsf{e}_i^{sid_i} = \mathsf{rcv}(t)$. This case is similar to the previous one.

Case $\mathsf{e}_i^{sid_i} = \mathsf{snd}(t)$. In such a case, we have that $\mathsf{ee}_i^{sid_i} = \mathsf{snd}(t')$ for some term t' such that $t = t'\sigma$. Let $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \mathsf{K}(\mathsf{exec}_i) \cup T_0))$. The only case for which we can not easily conclude by applying our induction hypothesis is when $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$. So, assume that

$u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$. According to Lemma 14, $u \in \mathsf{comp}(t'\sigma) \cup \bigcup_{x \in vars(t')} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$ where $\tau = \mathsf{ExpectedTag}(\mathsf{exec}, sid_i)$. We distinguish two cases:

(1) $u \in \mathsf{comp}(t'\sigma)$. We have that $t'\sigma = t \in \mathsf{K}(\mathsf{exec}_i)$ and thus $\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash u$.

(2) $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$ for some $x \in vars(t')$ and $u \notin \mathsf{comp}(t'\sigma)$. By the origination property we know that there exists $j < i$ such that $sid_j = sid_i$, $\mathsf{ee}_j^{sid_j} = \mathsf{rcv}(v')$ with $x \in vars(v')$, and thus that $x\sigma \in \mathsf{St}(v'\sigma)$. By Lemma 15, we deduce that $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, v'\sigma)$, and thanks to Lemma 11 (Item 2), we have that $u \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, v'\sigma)$. We can then apply our induction hypothesis in order to deduce that $\mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0 \vdash w$ for any $w \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0)$, and because exec is a valid trace, we have also that $\mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0 \vdash v'\sigma$. Thus, according Lemma 13, we deduce that $\mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0 \vdash w$ for any $w \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, v'\sigma)$. In particular, we conclude that $\mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0 \vdash u$. □

Appendix C. Proofs of section 6.3

In order to show the validity of the resulting trace, we first characterize the subterms that are abstracted by our transformation. Actually, we can show that those subterms are alien subterms, and thus they enjoy the properties established in Appendix B.

Lemma 17. Let Π be a k-party protocol, exec be an execution trace of $\widetilde{\Pi}$, t be a term and p be a position. If there exists sid such that $(\overline{t}^{\mathsf{exec},sid})|_p \in \mathcal{N}_\epsilon$, then we have that $t|_p \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$.

Proof. We will prove by induction on p that $t|_p \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$.

Base case $p = \epsilon$. In that case, according to Definition 19, either $t \in \mathcal{N}$, or $t = \mathsf{f}(t_1, \ldots, t_n)$ for some $\mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$. If $t \in \mathcal{N}$, we have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t) = \{t\}$, and thus $t|_p = t|_\epsilon \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$. Otherwise, i.e. $t = \mathsf{f}(t_1, \ldots, t_n)$ for some $\mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$, then we have that

$$\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t) = \{t\} \cup \bigcup_{i \in \{1,\ldots,n\}} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, t_i)$$

where $\tau = \mathsf{HeadTag}(\mathsf{exec}, t)$. Hence, we have that $t|_p = t|_\epsilon \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$.

Inductive case $p = i_0 \cdot q$. First, note that t cannot be a long-term key, i.e. t is not a term of the form $\mathsf{pub}(t')$, $\mathsf{priv}(t')$ or $\mathsf{shk}(t_1, t_2)$. Indeed, in such a case, $(\overline{t}^{\mathsf{exec},sid})|_p \notin \mathcal{N}_\epsilon$ for any sid. This would contradict one of our hypothesis. Thus, two cases remain:

Case $t = \langle t_1, t_2 \rangle$**.** By Definition 19, $\overline{t}^{\mathsf{exec},sid} = \langle \overline{t_1}^{\mathsf{exec},sid}, \overline{t_2}^{\mathsf{exec},sid} \rangle$. Suppose $i_0 = 1$. Then $(\overline{t_1}^{\mathsf{exec},sid})|_q \in \mathcal{N}_\epsilon$, and thanks to our induction hypothesis we can derive that $t_1|_q \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t_1)$. We have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t) = \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t_1) \cup \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t_2)$, thus $t_1|_q \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$. Finally, since $t_1|_q = t|_{1 \cdot q} = t|_p$ we can conclude that $t|_p \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$. The case where $i_0 = 2$ can be done in a similar way.

Case $t = \mathsf{f}(t_1, \ldots, t_n)$ **for some** $\mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$**.** Since $(\overline{t}^{\mathsf{exec},sid})|_{i_0 \cdot q} \in \mathcal{N}_\epsilon$, we know that $(\overline{t}^{\mathsf{exec},sid})|_\epsilon \notin \mathcal{N}_\epsilon$. Hence, by Definition 19, we have that

- $\overline{t}^{\mathsf{exec},sid} = f(\overline{t_1}^{\mathsf{exec},sid}, \ldots, \overline{t_n}^{\mathsf{exec},sid})$;
- $\mathsf{HeadTag}(\mathsf{exec}, t) = \mathsf{ExpectedTag}(\mathsf{exec}, sid) \neq \bot$; and

- $(\overline{t}^{\mathsf{exec},sid})|_{i_0.q} = (\overline{t_{i_0}}^{\mathsf{exec},sid})|_q$.

Hence, we have that $(\overline{t_{i_0}}^{\mathsf{exec},sid})|_q \in \mathcal{N}_\epsilon$. Thanks to our induction hypothesis, we deduce that $t_{i_0}|_q \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t_{i_0})$. Applying Lemma 12, we conclude that $t_{i_0}|_q \in \mathsf{comp}(t_{i_0}) \cup \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$.

In order to conclude, it is sufficient to show that if $t_{i_0}|_q \in \mathsf{comp}(t_{i_0})$, then we also have that $t_{i_0}|_q \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$. Assume that $t_{i_0}|_q \in \mathsf{comp}(t_{i_0})$. First, let $\tau = \mathsf{HeadTag}(\mathsf{exec}, t)$, thanks to Lemma 11 (item 1), we have that:

$$\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t) = \{t\} \cup \bigcup_{j\in\{1,\ldots,n\}} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, t_j) = \{t\} \cup \bigcup_{i=1}^{n} \bigcup_{t'_j \in \mathsf{comp}(t_j)} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, t'_j)$$

Hence, we have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, (t_{i_0})|_q) \subseteq \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, t)$. To conclude, it is hence enough to show that $t_{i_0}|_q \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, t_{i_0}|_q)$. Since $(\overline{t}^{\mathsf{exec},sid})|_p = (\overline{t}^{\mathsf{exec},sid})|_{i_0.q} = (\overline{t_{i_0}}^{\mathsf{exec},sid})|_q \in \mathcal{N}_\epsilon$, we need to distinguish three cases:

- *Case* $(t_{i_0})|_q \in \mathcal{N}_\epsilon$. In such a case, we have that $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, (t_{i_0})|_q) = \{(t_{i_0})|_q)\}$.
- *Case* $(t_{i_0})|_q \in \mathcal{N} \setminus \mathcal{N}_\epsilon$. In such a case, we have that $(t_{i_0})|_q = n_y^{sid'}$ for some $sid' \notin \mathsf{sameTagAs}(\mathsf{exec}, sid)$, thus $\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, (t_{i_0})|_q) = \{(t_{i_0})|_q)\}$.
- *Case* $(t_{i_0})|_q = \mathsf{g}(u_1, \ldots, u_m)$ *for some* $\mathsf{g} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$. In such a case, we have that $\mathsf{HeadTag}(t_{i_0}|_q) \neq \tau$, and thus
$$\mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, (t_{i_0})|_q) = \{(t_{i_0})|_q)\} \cup \bigcup_{j\in\{1,\ldots,m\}} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \mathsf{HeadTag}(t_{i_0}|_q), u_j). \qquad \square$$

We show that our transformation preserves disequalities even if terms are not abstracted using the same session identifier. This result can be proved by structural induction on $\overline{m}^{\mathsf{exec},sid}$.

Lemma 18. Let Π be a k-party protocol and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a valid execution trace of $\widetilde{\Pi}$, w.r.t. some initial intruder knowledge T_0. Let m and m' be two terms such that $m \neq m'$, and sid, sid' be two session identifiers. We have that $\overline{m}^{\mathsf{exec},sid} \neq \overline{m'}^{\mathsf{exec},sid'}$.

Lemma 4. Let Π be a k-party protocol and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a valid execution trace of $\widetilde{\Pi}$, w.r.t. some set T_0 of ground atoms. Let $i \in \{0, \ldots, \ell\}$ and t be a term such that $\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t$. We have that $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid}$ for any sid.

Proof. Let $\mathsf{tr} = [\mathsf{ee}_1^{sid_1}; \ldots; \mathsf{ee}_\ell^{sid_\ell}]$ be the symbolic trace associated to exec and σ be the substitution such that $\mathrm{dom}(\sigma) = vars(\mathsf{tr})$ and $\mathsf{exec} = \mathsf{tr}\sigma$. Let $i \in \{0, \ldots, \ell\}$. Let π be a simple proof of $\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t$. We prove that $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid}$ by induction on (i, π). If $i = 0$ and π is a simple proof reduced to a leaf (possibly followed by some projection rules), then we have that $T_0 \vdash t$, and π is necessarily reduced to a leaf since T_0 only contains atomic terms. Let sid be a session identifier, we have that $\overline{t}^{\mathsf{exec},sid} \in \{t\} \cup \mathcal{N}_\epsilon$. This allows us to conclude that $T_0 \vdash \overline{t}^{\mathsf{exec},sid}$ Now, we distinguish several cases depending on the last rule of π.

The proof π ends with an instance of a composition rule, i.e. $t = \mathsf{f}(t_1, \ldots, t_n)$ for some $\mathsf{f} \in \{\langle,\rangle, \mathsf{encs}, \mathsf{enca}, \mathsf{sign}, \mathsf{h}\}$ and some terms $t_1, \ldots, t_n$.

According to Definition 19, we have that $\overline{t}^{\mathsf{exec},sid} \in \mathcal{N}_\epsilon \cup \{\mathsf{f}(\overline{t_1}^{\mathsf{exec},sid}, \ldots, \overline{t_n}^{\mathsf{exec},sid})\}$. If $\overline{t}^{\mathsf{exec},sid} \in \mathcal{N}_\epsilon$, we easily conclude that $\mathsf{K}(\overline{\mathsf{exec}}_i) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid}$. Otherwise, since π ends with

a composition rule, we have that $\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t_1, \ldots, \mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t_n$. Moreover, the simple proofs witnessing these facts are strict subproofs of π that are also simple. Hence, we can apply our induction hypothesis in order to conclude that $\mathsf{K}(\overline{\mathsf{exec}_i}) \cup T_0 \vdash \overline{t_1}^{\mathsf{exec},sid}, \ldots, \mathsf{K}(\overline{\mathsf{exec}_i}) \cup T_0 \vdash \overline{t_n}^{\mathsf{exec},sid}$. This allows us to conclude that $\mathsf{K}(\overline{\mathsf{exec}_i}) \cup T_0 \vdash \mathsf{f}(\overline{t_1}^{\mathsf{exec},sid}, \ldots, \overline{t_n}^{\mathsf{exec},sid})$.

The proof ends with the application of a decomposition rule (but not a projection) possibly followed by several applications of the projection rules until the resulting term is not a pair. We will here present the case of the symmetric decryption rule, but all the other decomposition rules (including the case where the proof is reduced to a leaf) can be handled in a similar way. For some terms t_1 and t_2, the proof π is of the form

$$\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash \mathsf{encs}(t_1,t_2)} \quad \dfrac{\vdots}{\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t_2}}{\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t_1}}{\dfrac{\vdots}{\mathsf{K}(\mathsf{exec}_i) \cup T_0 \vdash t}}$$

Let us first note that, by locality (Lemma 3) and by simplicity of π we know that $\mathsf{encs}(t_1,t_2) \in \mathsf{St}(\mathsf{K}(\mathsf{exec}_i)) \cup T_0 \cup \mathcal{K}_\epsilon \cup \mathcal{N}_\epsilon \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\}$, and by atomicity of T_0, $\mathcal{N}_\epsilon$, $\mathcal{K}_\epsilon$ and $\{\mathsf{pub}(a) \mid a \in \mathcal{A}\}$, we know that $\mathsf{encs}(t_1,t_2) \in \mathsf{St}(\mathsf{K}(\mathsf{exec}))$. (In case of a proof reduced to a leaf, and if there is no projection rule, we may have that $t \in T_0$. In such a case, as in the base case, we have that $T_0 \vdash \overline{t}^{\mathsf{exec},sid}$ and we easily conclude.) Hence, there exists $k \leq i$ such that $\mathsf{e}_k^{sid_k} = \mathsf{snd}(u)$ and $\mathsf{encs}(t_1,t_2) \in \mathsf{St}(u)$. Let k_0 be the smallest such k and u_0, u_0' be such that $\mathsf{e}_{k_0}^{sid_{k_0}} = \mathsf{snd}(u_0)$ and $\mathsf{ee}_{k_0}^{sid_{k_0}} = \mathsf{snd}(u_0')$. Hence, we have that $u_0 = u_0'\sigma$.

In order to prove the result, we first establish the following claim.

Claim: We have that $\overline{\mathsf{encs}(t_1,t_2)}^{\mathsf{exec},sid_{k_0}} = \mathsf{encs}(\overline{t_1}^{\mathsf{exec},sid_{k_0}}, \overline{t_2}^{\mathsf{exec},sid_{k_0}})$.

Assume by contradiction, that this equality does not hold.

First, we have that $\mathsf{encs}(\overline{t_1}^{\mathsf{exec},sid_{k_0}}, \overline{t_2}^{\mathsf{exec},sid_{k_0}}) \notin \mathsf{St}(\overline{u_0}^{\mathsf{exec},sid_{k_0}})$. Indeed, for having $\mathsf{encs}(\overline{t_1}^{\mathsf{exec},sid_{k_0}}, \overline{t_2}^{\mathsf{exec},sid_{k_0}}) \in \mathsf{St}(\overline{u_0}^{\mathsf{exec},sid_{k_0}})$, there must exist $v \in \mathsf{St}(u_0)$ such that $\overline{v}^{\mathsf{exec},sid_{k_0}} = \mathsf{encs}(\overline{t_1}^{\mathsf{exec},sid_{k_0}}, \overline{t_2}^{\mathsf{exec},sid_{k_0}})$. But this would imply that $v = \mathsf{encs}(t_1', t_2')$ for some terms t_1', t_2' such that $\overline{t_1'}^{\mathsf{exec},sid_{k_0}} = \overline{t_1}^{\mathsf{exec},sid_{k_0}}$ and $\overline{t_2'}^{\mathsf{exec},sid_{k_0}} = \overline{t_2}^{\mathsf{exec},sid_{k_0}}$. However this would in turn imply according to Lemma 18 that $t_1' = t_1$ and $t_2' = t_2$. In other words we would have $v = \mathsf{encs}(t_1,t_2) \in \mathsf{St}(u_0)$ but with $\overline{v}^{\mathsf{exec},sid_{k_0}} = \mathsf{encs}(\overline{t_1}^{\mathsf{exec},sid_{k_0}}, \overline{t_2}^{\mathsf{exec},sid_{k_0}})$ which would contradict our hypothesis. Hence, necessarily $\mathsf{encs}(\overline{t_1}^{\mathsf{exec},sid_{k_0}}, \overline{t_2}^{\mathsf{exec},sid_{k_0}}) \notin \mathsf{St}(\overline{u_0}^{\mathsf{exec},sid_{k_0}})$.

Now since $\mathsf{encs}(\overline{t_1}^{\mathsf{exec},sid_{k_0}}, \overline{t_2}^{\mathsf{exec},sid_{k_0}}) \notin \mathsf{St}(\overline{u_0}^{\mathsf{exec},sid_{k_0}})$, while $\mathsf{encs}(t_1,t_2) \in \mathsf{St}(u_0)$, there must exist a position p (smaller or equal to the position where $\mathsf{encs}(t_1,t_2)$ occurs in u_0) such that $(\overline{u_0}^{\mathsf{exec},sid_{k_0}})|_p \in \mathcal{N}_\epsilon$ and $\mathsf{encs}(t_1,t_2) = u_0|_p$. Hence, Lemma 17 tells us that $u_0|_p \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, u_0)$. Thanks to Lemma 14 and Lemma 11 (Item 2), we conclude that:

$$u_0|_p \in \mathsf{comp}(u_0) \cup \bigcup_{x \in vars(u_0')} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, x\sigma)$$

We now distinguish two cases and show that each case leads us to a contradiction.
Case 1: $u_0|_p \in \mathsf{comp}(u_0) \setminus \bigcup_{x \in vars(u_0')} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, x\sigma)$. In such a case, there exists $u_0'' \in \mathsf{comp}(u_0')$ such that $u_0|_p \in \mathsf{comp}(u_0''\sigma)$. But because we are considering the case where $u_0|_p \notin \bigcup_{x \in vars(u_0')} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, x\sigma)$, it must be that $u_0|_p = u_0''\sigma$. Now, by construction of tr, it must be that $\mathsf{ExpectedTag}(\mathsf{exec}, sid_{k_0}) = \mathsf{HeadTag}(\mathsf{exec}, u_0''\sigma)$, and thus $\overline{(u_0)|_p}^{\mathsf{exec},sid_{k_0}} \notin \mathcal{N}_\epsilon$. Finally, because $(u_0)|_p \in \mathsf{comp}(u_0)$, we have that $(\overline{u_0}^{\mathsf{exec},sid_{k_0}})|_p = \overline{(u_0)|_p}^{\mathsf{exec},sid_{k_0}}$. However, this equality is not possible since we have shown that $(\overline{u_0}^{\mathsf{exec},sid_{k_0}})|_p \in \mathcal{N}_\epsilon$ whereas $\overline{(u_0)|_p}^{\mathsf{exec},sid_{k_0}} \notin \mathcal{N}_\epsilon$. Hence, we obtain a contradiction.
Case 2: $u_0|_p \in \bigcup_{x \in vars(u_0')} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, x\sigma)$. In such a case, there exists $x \in vars(u_0')$ such that $u_0|_p \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, x\sigma)$ and $\mathsf{encs}(t_1, t_2) \in \mathsf{St}(x\sigma)$. Thanks to the origination property (see Definition 3 - Condition 1), we know that there exists $j < k_0$ such that $\mathsf{ee}_j^{sid_j} = \mathsf{rcv}(v')$ and $x \in vars(v')$. Hence, we have that $\mathsf{encs}(t_1, t_2) \in \mathsf{St}(v'\sigma)$. Since exec is a valid trace, we have that $\mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0 \vdash v'\sigma$.

Let π' be a simple proof of $\mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0 \vdash v'\sigma$, and π'' be a minimal subproof of π' whose root is labeled with a term t' such that $\mathsf{encs}(t_1, t_2) \in \mathsf{St}(t')$. By locality of π' (Lemma 3), and because $\mathsf{encs}(t_1, t_2) \notin \mathsf{St}(\mathsf{K}(\mathsf{exec}_{j-1}))$ (remember here that we choose k_0 such that for all $j < k_0$, we have that $\mathsf{encs}(t_1, t_2) \notin \mathsf{St}(\mathsf{K}(\mathsf{exec}_{j-1}))$), we know that π'' ends with a composition rule. Unless $t' = \mathsf{encs}(t_1, t_2)$, this contradicts the minimality of π''. Hence, we have that $t' = \mathsf{encs}(t_1, t_2)$ and π'' is a simple proof of $\mathsf{encs}(t_1, t_2)$ whose last rule is a composition. Actually, since $\mathsf{encs}(t_1, t_2) \notin \mathsf{St}(\mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0)$, any simple proof of $\mathsf{encs}(t_1, t_2)$ ends with a composition. This will contradict the fact that π is a simple proof of t.

This allows us to conclude the proof of the claim.

Now, by relying on our claim and by applying our induction hypothesis, we have that:

- $\mathsf{K}(\overline{\mathsf{exec}_i}) \cup T_0 \vdash \mathsf{encs}(\overline{t_1}^{\mathsf{exec},sid_{k_0}}, \overline{t_2}^{\mathsf{exec},sid_{k_0}})$; and
- $\mathsf{K}(\overline{\mathsf{exec}_i}) \cup T_0 \vdash \overline{t_2}^{\mathsf{exec},sid_{k_0}}$.

This allows us to deduce that $\mathsf{K}(\overline{\mathsf{exec}_i}) \cup T_0 \vdash \overline{t_1}^{\mathsf{exec},sid_{k_0}}$.

In order to establish that $\mathsf{K}(\overline{\mathsf{exec}_i}) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid}$, we need to distinguish several cases:

Case $t \in \mathcal{A}$, $t = \mathsf{pub}(a)$ or $t = \mathsf{f}(a_1, \ldots, a_n)$ for some $\mathsf{f} \in \{\mathsf{shk}, \mathsf{priv}\}$:
In such a case, we have that $\overline{t}^{\mathsf{exec},sid} = \overline{t}^{\mathsf{exec},sid_{k_0}} = t$. Hence, we have that $\mathsf{K}(\overline{\mathsf{exec}_i}) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid}$ by applying some projection rules on the proof of $\mathsf{K}(\overline{\mathsf{exec}_i}) \cup T_0 \vdash \overline{t_1}^{\mathsf{exec},sid_{k_0}}$.

Case $t \in \mathcal{N}$ or $t = \mathsf{f}(t_1', \ldots, t_m')$ for some $\mathsf{f} \in \{\mathsf{encs}, \mathsf{enca}, \mathsf{h}, \mathsf{sign}\}$:
If $\overline{t}^{\mathsf{exec},sid} \in \mathsf{comp}(\overline{t_1}^{\mathsf{exec},sid_{k_0}})$, then we easily conclude that $\mathsf{K}(\overline{\mathsf{exec}_i}) \cup T_0 \vdash \overline{t}^{\mathsf{exec},sid}$ since we have established that $\mathsf{K}(\overline{\mathsf{exec}_i}) \cup T_0 \vdash \overline{t_1}^{\mathsf{exec},sid_{k_0}}$. Otherwise, we have that $\overline{t}^{\mathsf{exec},sid} \notin \mathsf{comp}(\overline{t_1}^{\mathsf{exec},sid_{k_0}})$. In that case, and according to Definition 19 and Lemma 11 (item 1), either $\overline{t}^{\mathsf{exec},sid} \in \mathcal{N}_\epsilon$ or $\overline{t}^{\mathsf{exec},sid_{k_0}} \in \mathcal{N}_\epsilon$. In the first case, we trivially conclude. In the second case, i.e. $\overline{t}^{\mathsf{exec},sid} \notin \mathcal{N}_\epsilon$ but $\overline{t}^{\mathsf{exec},sid_{k_0}} \in \mathcal{N}_\epsilon$, we have that $t \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \mathsf{encs}(t_1, t_2))$ (thanks to Lemma 17. Since $\mathsf{encs}(t_1, t_2) \in \mathsf{St}(u_0)$, we deduce that $t \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, u_0) \cup \mathsf{comp}(\mathsf{encs}(t_1, t_2))$ by applying Lemma 12. Now,

since $t \neq \mathsf{encs}(t_1, t_2)$, we deduce that $t \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, u_0)$. Thus, applying Lemma 14, we have that

$$t \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, u_0) \subseteq \mathsf{comp}(u_0) \cup \bigcup_{x \in vars(u_0')} \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$$

where $\tau = \mathsf{ExpectedTag}(\mathsf{exec}, sid_{k_0})$.

Assume that $t \in \mathsf{comp}(u_0)$ and $t \notin \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$ for any $x \in vars(u_0')$. In such a case, we have that there exists $t' \in \mathsf{comp}(u_0')$ such that $t \in \mathsf{comp}(t'\sigma)$ and we know that $t' \notin vars(u_0')$. Hence t is either a nonce and we have that $t = t'$. Moreover, we know that $t' = {n_y}^{sid_{k_0}}$ for some y (by construction of tr. In such a case, $\overline{t}^{\mathsf{exec}, sid_{k_0}} \notin \mathcal{N}_\epsilon$. This leads us to a contradiction. Otherwise t is an encrypted term and we have that $t = t'\sigma$ and again by construction of tr, we have that $\overline{t'\sigma}^{\mathsf{exec}, sid_{k_0}} \notin \mathcal{N}_\epsilon$, leading us to a contradiction. Hence, we know that this case is not possible.

Hence, we have that $t \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, x\sigma)$ for some $x \in vars(u_0')$. Thanks to the origination property, we know that there exists $j < k_0$ such that $sid_j = sid_{k_0}$, $\mathsf{ee}_j^{sid_j} = \mathsf{rcv}(v')$ with $x \in vars(v')$. Hence, we have that $x\sigma \in \mathsf{St}(v'\sigma)$. Then, applying Lemma 15, we deduce that $t \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \tau, v'\sigma)$, and thanks to Lemma 11 (item 2), we have that $t \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, v'\sigma)$.

Now, according to Lemma 16, we know that $\mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0 \vdash w$ for any $w \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, \mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0)$. Since exec is a valid trace, we have that $\mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0 \vdash v'\sigma$. Applying Lemma 13, we deduce that $\mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0 \vdash w$ for any $w \in \mathsf{St}_{\mathsf{alien}}(\mathsf{exec}, v)$. In particular, we have that $\mathsf{K}(\mathsf{exec}_{j-1}) \cup T_0 \vdash t$ and we conclude by relying on our induction hypothesis. □

Appendix D. Proofs of Section 6.4

In order to prove Proposition 3 we will annotate formulas. For the sake of homogeneity, we chose to annotate each term that occurs in the formula even though it would have been sufficient to only annotate variables. Moreover, we state the definition for a general formula, but in our setting, terms that occur in a formula are either names or variables.

Definition 27. (annotated formula) Given a formula ϕ, we define its annotated version $\mathsf{annotate}(\phi)$ as follows:

$\mathsf{annotate}(\mathsf{true}) = \mathsf{true}$	$\mathsf{annotate}(\mathsf{Q}(t_1, \ldots, t_n)) = \mathsf{Q}(t_1^{t_1}, \ldots, t_n^{t_n})$
$\mathsf{annotate}(\neg\phi) = \neg\mathsf{annotate}(\phi)$	$\mathsf{annotate}(\phi_1 \vee \phi_2) = \mathsf{annotate}(\phi_1) \vee \mathsf{annotate}(\phi_2)$
$\mathsf{annotate}(\mathsf{learn}(t)) = \mathsf{learn}(t^t)$	$\mathsf{annotate}(\Diamond\phi) = \Diamond\mathsf{annotate}(\phi)$
$\mathsf{annotate}(\mathsf{C}(u)) = \mathsf{C}(u^u)$	$\mathsf{annotate}(\exists x.\phi) = \exists x.\mathsf{annotate}(\phi)$

We emphasize that those annotations are syntactic decorations that do not interfere in the semantics of the formulas. We also suppose that these annotations are not affected by substitutions, i.e., when x is a variable annotated with a, $(x^a)\sigma = (x\sigma)^a$. Relying on this notion of annotated formulas, we are now able to link each variable that occurs in ϕ with the term it has been substituted with in order to satisfy the formula. More precisely, we only need to know the session identifiers from which those terms are issued. The idea is that these sessions are important to satisfy the attack formula whereas the other ones could be discarded from the execution trace.

Definition 28. Let ϕ be an attack formula and ψ its annotated version, i.e. $\psi = \mathsf{annotate}(\phi)$. Let Π be a protocol, and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be an execution trace (not necessarily valid) of Π w.r.t. some initial intruder knowledge T_0 and such that $\langle \mathsf{exec}, T_0 \rangle \models \psi$. Let π be a proof tree witnessing the fact that $\langle \mathsf{exec}, T_0 \rangle \models \psi$. We define $\mu(\pi)$ as described in Figure 2.

$$\mu\left(\frac{}{\langle \mathsf{exec}_i, T_0 \rangle \models \mathsf{true}}\right) = \emptyset \qquad \mu\left(\frac{}{\langle \mathsf{exec}_i, T_0 \rangle \models \mathsf{learn}(t^u)}\right) = \emptyset$$

$$\mu\left(\frac{}{\langle \mathsf{exec}_i, T_0 \rangle \models \mathsf{C}(u^v)}\right) = \emptyset \qquad \mu\left(\frac{}{\langle \mathsf{exec}_i, T_0 \rangle \models \neg\mathsf{C}(u^v)}\right) = \emptyset$$

$$\mu\left(\frac{}{\langle \mathsf{exec}_i, T_0 \rangle \models \neg\mathsf{Q}(t_1^{u_1}, \ldots, t_n^{u_n})}\right) = \emptyset$$

$$\mu\left(\frac{}{\langle \mathsf{exec}_i, T_0 \rangle \models \mathsf{Q}(t_1^{u_1}, \ldots, t_n^{u_n})}\right) = \{(u'_1, sid_i); ..., (u'_m; sid_i)\} \text{ where } vars(\{u_1, \ldots, u_n\}) = \{u'_1, \ldots, u'_m\}$$

$$\mu\left(\frac{\dfrac{\pi'}{\langle \mathsf{exec}_i, T_0 \rangle \models \psi_j}}{\langle \mathsf{exec}_i, T_0 \rangle \models \psi_1 \vee \psi_2}\right) = \mu\left(\frac{\pi'}{\langle \mathsf{exec}_i, T_0 \rangle \models \psi_j}\right) \text{ with } j \in \{1, 2\}$$

$$\mu\left(\frac{\dfrac{\pi_1}{\langle \mathsf{exec}_i, T_0 \rangle \models \neg\psi_1} \quad \dfrac{\pi_2}{\langle \mathsf{exec}_i, T_0 \rangle \models \neg\psi_2}}{\langle \mathsf{exec}_i, T_0 \rangle \models \neg(\psi_1 \vee \psi_2)}\right) = \bigcup_{j \in \{1,2\}} \mu\left(\frac{\pi_j}{\langle \mathsf{exec}_i, T_0 \rangle \models \neg\psi_j}\right)$$

$$\mu\left(\frac{\dfrac{\pi'}{\langle \mathsf{exec}_j, T_0 \rangle \models \psi}}{\langle \mathsf{exec}_i, T_0 \rangle \models \Diamond\psi}\right) = \mu\left(\frac{\pi'}{\langle \mathsf{exec}_j, T_0 \rangle \models \psi}\right) \text{ where } j \leq i$$

$$\mu\left(\frac{\dfrac{\pi_1}{\langle \mathsf{exec}_1, T_0 \rangle \models \neg\psi} \quad \ldots \quad \dfrac{\pi_i}{\langle \mathsf{exec}_i, T_0 \rangle \models \neg\psi}}{\langle \mathsf{exec}_i, T_0 \rangle \models \neg(\Diamond\psi)}\right) = \bigcup_{i \in \{1,\ldots,i\}} \mu\left(\frac{\pi_i}{\langle \mathsf{exec}_i, T_0 \rangle \models \neg\psi}\right)$$

$$\mu\left(\frac{\dfrac{\pi'}{\langle \mathsf{exec}_i, T_0 \rangle \models \psi\{t/x\}}}{\langle \mathsf{exec}_i, T_0 \rangle \models \exists x.\psi}\right) = \mu\left(\frac{\pi'}{\langle \mathsf{exec}_i, T_0 \rangle \models \psi\{t/x\}}\right)$$

$$\mu\left(\frac{\dfrac{\pi'}{\langle \mathsf{exec}_i, T_0 \rangle \models \psi}}{\langle \mathsf{exec}_i, T_0 \rangle \models \neg\neg\psi}\right) = \mu\left(\frac{\pi'}{\langle \mathsf{exec}_i, T_0 \rangle \models \psi}\right)$$

Figure 2: Definition of the function μ

Intuitively, $\mu(\pi)$ maps variables occurring positively in a status event in the attack formula ϕ to session identifiers. Note also that since by definition of an attack formula each variable occurs at most once in a positive status event and by Condition 4 of Definition 10, we have that $\mu(\pi)$ is actually a function.

Proposition 3. Let Π be a protocol, exec be an execution trace of $\widetilde{\Pi}$ w.r.t. some initial intruder knowledge T_0, and ϕ be an attack formula. We have that

$$\langle \mathsf{exec}, T_0 \rangle \models \phi \;\Rightarrow\; \langle \overline{\mathsf{exec}}, T_0 \rangle \models \phi.$$

Proof. Let $\mathsf{exec} = [\mathsf{e}_1^{sid_1}, \ldots, \mathsf{e}_\ell^{sid_\ell}]$ for some ℓ, and some session identifiers $sid_1, \ldots, sid_\ell$. By definition of an attack formula, ϕ is of the form

$$\phi = \exists x_1. \ldots . \exists x_n . \psi$$

for some quantifier-free formula ψ. Now, according to the semantics of $\mathcal{L}$, $\langle \mathsf{exec}, T_0 \rangle \models \phi$ implies that there exists n ground terms $m_1, \ldots, m_n$ such that there exists a proof π of $\langle \mathsf{exec}, T_0 \rangle \models \phi$ of the form:

$$\pi \;=\; \dfrac{\dfrac{\ldots}{\langle \mathsf{exec}, T_0 \rangle \models \psi^a \sigma}}{\langle \mathsf{exec}, T_0 \rangle \models \phi^a}$$

where $\sigma = \{x_1 \mapsto m_1, \ldots, x_n \mapsto m_n\}$ and $\phi^a = \exists x_1. \ldots . \exists x_n . \psi^a = \mathsf{annotate}(\phi)$. Let $\overline{\sigma} = \{x_1 \mapsto \overline{m_1}^{\mathsf{exec},sid'_1}, \ldots, x_n \mapsto \overline{m_n}^{\mathsf{exec},sid'_n}\}$ where $sid'_j = \mu(\pi)(x_j)$ when $x_j \in \mathrm{dom}(\mu(\pi))$ and 0 otherwise.

Note that all except the last two nodes of π are labeled with $\langle \mathsf{exec}_i, T_0 \rangle \models \psi' \sigma$ where $i \leq \mathsf{length}(\mathsf{exec})$ and ψ' is smaller than ψ. Thus, the proof tree is finite. Moreover, by definition of μ, we have that any leaf of π of the form $\langle \mathsf{exec}_i, T_0 \rangle \models \mathsf{Q}(u_1, \ldots, u_k)\sigma$ is such that $\mu(\pi)(x) = sid_i$ for any $x \in vars(\{u_1, \ldots, u_k\})$. We prove that the proof tree obtained from π by replacing each node labeled with $\langle \mathsf{exec}_i, T_0 \rangle \models \psi' \sigma$ by $\langle \mathsf{exec}_i, T_0 \rangle \models \psi' \overline{\sigma}$ is a (valid) proof tree witnessing the fact that $\langle \overline{\mathsf{exec}}, T_0 \rangle \models \psi^a \overline{\sigma}$.

Base cases: the leaves of the proof tree π. In such a case, we have $\langle \mathsf{exec}_i, T_0 \rangle \models \psi_0 \sigma$ for a formula ψ_0 of the form true, $\mathsf{C}(x)$, $\neg\mathsf{C}(x)$, $\mathsf{learn}(u_0)$, $\mathsf{Q}(u_1, \ldots, u_k)$, or $\neg Q(u_1, \ldots, u_k)$.

- $\psi_0 = \mathsf{true}$: in such a case, we easily conclude.
- $\psi_0 = \mathsf{C}(x)$ (resp. $\neg\mathsf{C}(x)$): in such a case, we have that $\mathsf{C}(x\sigma) = \mathsf{C}(x\overline{\sigma})$ since $\overline{a}^{\mathsf{exec},sid} = a$ for any agent name a and any sid, and since the semantics of C does not rely on the execution trace, we can also easily conclude in this case.
- $\psi_0 = \mathsf{learn}(u_0)$: in such a case, by definition of an attack formula, we know that u_0 is either an agent name (in such a case, we easily conclude) or a variable in $\{x_1, \ldots, x_n\}$. Let j be such that $u_0 = x_j$. By hypothesis, we have that $T_0 \cup \mathsf{K}(\mathsf{exec}_i) \vdash u_0\sigma$. According to Lemma 4, we know that $T_0 \cup \mathsf{K}(\overline{\mathsf{exec}}_i) \vdash \overline{u_0\sigma}^{\mathsf{exec},sid}$ for any sid, and thus in particular for sid'_j. Actually, we have that $\overline{x_j\sigma}^{\mathsf{exec},sid'_j} = x_j\overline{\sigma} (= \overline{m_j}^{\mathsf{exec},sid'_j})$, and this allows us to conclude that $\langle \overline{\mathsf{exec}}_i, T_0 \rangle \models \mathsf{learn}(u_0)\overline{\sigma}$.
- $\psi_0 = \mathsf{Q}(u_1, \ldots, u_k)$: in such a case, we know that each u_j is either an agent name or a variable, and we have that $u_j\sigma = t_j$ for any $j \in \{1, \ldots, k\}$ where $\mathsf{e}_i^{sid_i} = Q(t_1, \ldots, t_k)$. By definition of μ, we have that either u_j is an agent name or u_j is a variable and $\mu(\pi)(u_j) = sid_i$. In order to conclude that $\langle \overline{\mathsf{exec}}_i, T_0 \rangle \models \mathsf{Q}(u_1, \ldots, u_k)\overline{\sigma}$, we have to show that $\overline{t_j}^{\mathsf{exec},sid_i} = u_j\overline{\sigma}$. Let $j \in \{1, \ldots, k\}$. By hypothesis, we have that $u_j\sigma = t_j$, and thus $\overline{u_j\sigma}^{\mathsf{exec},sid_i} = \overline{t_j}^{\mathsf{exec},sid_i}$. We distinguish two cases. Either

u_j is an agent name, and we have that $\overline{u_j\sigma}^{\mathsf{exec},sid_i} = u_j = u_j\overline{\sigma}$. Otherwise, u_j is a variable, and we also have that $\overline{u_j\sigma}^{\mathsf{exec},sid_i} = u_j\overline{\sigma}$ since by definition of μ, we have that $\mu(\pi)(u_j) = sid_i$.

- $\psi_0 = \neg\mathsf{Q}(u_1,\ldots,u_k)$: in such a case, we know that each u_j is either an agent name or a variable, and we have that either $\mathsf{exec}_i = []$ or $\mathsf{Q}(u_1,\ldots,u_n)\sigma \neq \mathsf{e}_i^{sid_i}$. In the first case, we have that $\overline{\mathsf{exec}}_i = []$ and we easily conclude. From now on, assume that $\mathsf{Q}(u_1,\ldots,u_n)\sigma \neq \mathsf{e}_i^{sid_i}$. If $\mathsf{e}_i^{sid_i} \neq \mathsf{Q}(t_1,\ldots,t_k)$ for any terms $t_1,\ldots,t_k$, then it is easy to see that $\overline{\mathsf{e}_i^{sid_i}}^{\mathsf{exec},sid_i} \neq \mathsf{Q}(u_1,\ldots,u_k)\overline{\sigma}$ and this allows us to conclude. Now, assume that $\mathsf{e}_i^{sid_i} = \mathsf{Q}(t_1,\ldots,t_k)$ for some terms $t_1,\ldots,t_k$. In such a case, there exists $j \in \{1,\ldots,k\}$ such that $u_j\sigma \neq t_j$. Using Lemma 18, we deduce that $\overline{u_j\sigma}^{\mathsf{exec},\mu(u_j)} \neq \overline{t_j}^{\mathsf{exec},sid_i}$, and by definition of μ we have that $u_j\overline{\sigma} = \overline{u_j\sigma}^{\mathsf{exec},\mu(u_j)}$. This allows us to conclude that $\mathsf{Q}(u_1,\ldots,u_k)\overline{\sigma} \neq \overline{\mathsf{Q}(t_1,\ldots,t_k)}^{\mathsf{exec},sid_i}$, and thus $\langle\overline{\mathsf{exec}}_i, T_0\rangle \models \psi_0\overline{\sigma}$.

Inductive cases. In such a case, we have that $\langle\mathsf{exec}_i, T_0\rangle \models \psi_0\sigma$ for a formula ψ_0 of the form $\neg\neg\psi_0'$, $\psi_1 \vee \psi_2$, $\neg(\psi_1 \vee \psi_2)$, $\Diamond\psi_0'$, or $\neg\Diamond\psi_0'$.

- $\psi_0 = \neg\neg\psi_0'$: in such a case, we have that $\langle\mathsf{exec}_i, T_0\rangle \models \psi_0'\sigma$, and using our induction hypothesis we conclude that $\langle\overline{\mathsf{exec}}_i, T_0\rangle \models \psi_0'\overline{\sigma}$, and thus $\langle\overline{\mathsf{exec}}_i, T_0\rangle \models \neg\neg\psi_0'\overline{\sigma} = \psi_0\overline{\sigma}$.
- $\psi_0 = \psi_1 \vee \psi_2$: in such a case, we have that $\langle\mathsf{exec}_i, T_0\rangle \models \psi_j\sigma$ for some $j \in \{1,2\}$, and using our induction hypothesis we conclude that $\langle\overline{\mathsf{exec}}_i, T_0\rangle \models \psi_j\overline{\sigma}$, and thus $\langle\overline{\mathsf{exec}}_i, T_0\rangle \models (\psi_1 \vee \psi_2)\overline{\sigma} = \psi_0\overline{\sigma}$.
- $\psi_0 = \neg(\psi_1 \vee \psi_2)$: in such a case, we have that $\langle\mathsf{exec}_i, T_0\rangle \models \neg\psi_j'\sigma$ with $j \in \{1,2\}$, and using our induction hypothesis we conclude that $\langle\overline{\mathsf{exec}}_i, T_0\rangle \models \neg\psi_j\overline{\sigma}$ with $j \in \{1,2\}$, and thus $\langle\overline{\mathsf{exec}}_i, T_0\rangle \models \neg(\psi_1 \vee \psi_2)\overline{\sigma} = \psi_0\overline{\sigma}$.
- $\psi_0 = \Diamond\psi_0'$: in such a case, we have that $\langle\mathsf{exec}_j, T_0\rangle \models \psi_0'\sigma$ for some $j \leq i$, and using our induction hypothesis, we conclude that $\langle\overline{\mathsf{exec}_j}, T_0\rangle \models \psi_0'\overline{\sigma}$, and thus $\langle\overline{\mathsf{exec}}_i, T_0\rangle \models \Diamond\psi_0'\overline{\sigma} = \psi_0\overline{\sigma}$.
- $\psi_0 = \neg\Diamond\psi_0'$: in such a case, we have that $\langle\mathsf{exec}_j, T_0\rangle \models \neg\psi_0'\sigma$ for any $j \in \{1,\ldots,j\}$, and using our induction hypothesis, we conclude that $\langle\overline{\mathsf{exec}_j}, T_0\rangle \models \neg\psi_0'\overline{\sigma}$, and thus $\langle\overline{\mathsf{exec}}_i, T_0\rangle \models \neg\Diamond\psi_0'\overline{\sigma} = \psi_0\overline{\sigma}$. □

Appendix E. Proofs of Section 7

This appendix contains the proofs of Section 7. Actually, Section E.1 contains the proofs related to the validity of the resulting trace $\mathsf{exec}|_S$ whereas Section E.2 contains those related to the satisfiability of the attack formula.

E.1. **Validity of the resulting trace.**

In order to preserve the validity of the resulting trace, it is important to show that sessions that are not tagged in the same way cannot share any name. This is the purpose of the following lemma.

Lemma 5. Let Π be a k-party protocol, and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a well-formed valid execution of $\widetilde{\Pi}$ w.r.t. some set T_0 of ground atoms. Let $sess_1$ and $sess_2$ be two session identifiers. We have that:

$$\mathsf{sameTagAs}(\mathsf{exec}, sess_1) \neq \mathsf{sameTagAs}(\mathsf{exec}, sess_2)$$
$$\text{implies}$$
$$names(\mathsf{exec}, sess_1) \cap names(\mathsf{exec}, sess_2) = \emptyset$$

where $names(\mathsf{exec}, sess) = \{u \mid u \in names(\mathsf{e}_j^{sid_j})$ for some $1 \leq j \leq \ell$ such that $sid_j = sess\}$.

Proof. Let $sess_1$ and $sess_2$ be two sessions and n be a name such that:

- $\mathsf{sameTagAs}(\mathsf{exec}, sess_1) \neq \mathsf{sameTagAs}(\mathsf{exec}, sess_2)$; and
- $n \in names(\mathsf{exec}, sess_1) \cap names(\mathsf{exec}, sess_2)$.

Let $S = \mathsf{sameTagAs}(\mathsf{exec}, sess_1)$. According to Condition 3 of well-formedness (Definition 17), $n \in names(\mathsf{exec}, sess_1)$ implies that either n is of the form $n_t^{\epsilon,S}$ or of the form n_t^{sid} for some term t and session identifier $sid \in S$. We treat these two cases separately:

Case $n = n_t^{\epsilon,S}$**:** According to Condition 3 of well-formedness (Definition 17), $n_t^{\epsilon,S} \in names(\mathsf{exec}, sess_2)$ implies that $S = \mathsf{sameTagAs}(\mathsf{exec}, sess_2)$. But this contradicts the hypothesis $\mathsf{sameTagAs}(\mathsf{exec}, sess_1) \neq \mathsf{sameTagAs}(\mathsf{exec}, sess_2)$.

Case $n = n_t^{sid}$**:** In that case, $sid \in S$ and $\mathsf{sameTagAs}(\mathsf{exec}, sid) = \mathsf{sameTagAs}(\mathsf{exec}, sess_1)$. Now, according to Condition 3 of well-formedness (Definition 17), we have that $n_t^{sid} \in names(\mathsf{exec}, sess_2)$ implies that $sid \in \mathsf{sameTagAs}(\mathsf{exec}, sess_2)$. However, this means that $\mathsf{sameTagAs}(\mathsf{exec}, sid) = \mathsf{sameTagAs}(\mathsf{exec}, sess_2)$ which contradicts our hypothesis.

By contradiction we conclude that $names(\mathsf{exec}, sess_1) \cap names(\mathsf{exec}, sess_2) = \emptyset$. □

Now, provided that S and t satisfy some conditions, we show that a term t that was deducible from exec will still be deducible from $\mathsf{exec}|_S$.

Lemma 6. Let Π be a k-party protocol, and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ a well-formed valid execution of $\widetilde{\Pi}$ w.r.t. some set T_0 of ground atoms, and such that $T_0 \cup \mathsf{K}(\mathsf{exec}) \not\vdash k$ for any $k \in lgKeys \setminus (\mathcal{K}_\epsilon \cup T_0)$ (exec does not reveal any long term keys). Let S be a set of sessions such that:

> for all session identifiers $sess_1$ and $sess_2$ such that $\mathsf{sameTagAs}(\mathsf{exec}, sess_1) = \mathsf{sameTagAs}(\mathsf{exec}, sess_2)$, we have that $sess_1 \in S$ if and only if $sess_2 \in S$.

For all term $t \in \mathsf{St}(\mathsf{exec}|_S)$ such that $T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t$, we have that $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t$.

Proof. Let $sid \in S$, $t \in \mathsf{St}(\mathsf{exec}, sid)$, and π be a simple proof of $T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t$. We prove this result by structural induction on π. But, we first need to establish the following preliminary result (still under the hypotheses stated in Lemma 6).

Claim. If $names(t) \subseteq \mathcal{N}_\epsilon$ then $T_0 \vdash t$.

Proof of the claim. Let us suppose that there exists $u \in \mathsf{CryptSt}(t)$. Because exec is well-formed, we know by Conditions 1 and 2 of well-formedness (Definition 17) that t is k-tagged and thus that $u = \mathsf{f}(\langle \tau, u_1 \rangle, \ldots, u_n)$ with $\tau = \mathsf{ExpectedTag}(\mathsf{exec}, sid) \neq \bot$. Now, according to the definition of a symbolic trace (Definition 5) and of our protocol transformation (Definition 12), we know that there exists $n_v^{sid} \in names(\tau) \subseteq names(u) \subseteq names(t)$, which

contradicts the hypothesis that $names(t) \subseteq \mathcal{N}_\epsilon$. Thus it must be that $\mathsf{CryptSt}(t) = \emptyset$, and hence, t must be a tuple of atoms, i.e. a tuple of terms in $\mathcal{A} \cup T_0 \cup \mathcal{N}_\epsilon \cup \mathcal{K}_\epsilon \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\} \cup \{\mathsf{priv}(a), \mathsf{shk}(a,b) \mid a, b \in \mathcal{A}\}$. Now, because we only consider executions that do not reveal any long-term decryption keys, we necessarily have that the atomic subterms of t are in $\mathcal{A} \cup T_0 \cup \mathcal{N}_\epsilon \cup \mathcal{K}_\epsilon \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\}$. This implies according to Definition 1, that any atomic subterm of t is deducible from T_0. Finally, since t is a tuple of deducible terms, t can be deduced by application of the pairing rule, and thus $T_0 \vdash t$.

We now proceed with our induction

Base case: π is reduced to a leaf: In that case, $t \in \mathcal{A} \cup T_0 \cup \mathcal{N}_\epsilon \cup \mathcal{K}_\epsilon \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\} \cup \mathsf{K}(\mathsf{exec})$. If $names(t) \subseteq \mathcal{N}_\epsilon$, then by the above claim we have that $T_0 \vdash t$, and thus $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t$. Let us now suppose that there exists $n_v^{sid'} \notin \mathcal{N}_\epsilon$. In that case, $n_v^{sid'} \in names(\mathsf{exec}, sid)$ and $t \in \mathsf{K}(\mathsf{exec})$, i.e. there exists $i \in \{1, \ldots, \ell\}$, such that $\mathsf{e}_i^{sid_i} = \mathsf{snd}(t)$. Thus, $n_v^{sid'} \in names(\mathsf{exec}, sid_i)$ and hence $names(\mathsf{exec}, sid) \cap names(\mathsf{exec}, sid_i) \neq \emptyset$. This, according to Lemma 5, implies that

$$\mathsf{sameTagAs}(\mathsf{exec}, sid_i) \;=\; \mathsf{sameTagAs}(\mathsf{exec}, sid)$$

By hypothesis on S, we have $sid_i \in S$, and by definition we have $\mathsf{e}_i^{sid_i} \in \mathsf{exec}|_S$, which implies that $t \in \mathsf{K}(\mathsf{exec}|_S)$. We can thus conclude that $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t$.

Inductive case: In that case we need to distinguish two cases according to the last rule applied in the proof π.

Case 1 – the last rule is a composition rule: We have that the term t is of the form $\mathsf{f}(t_1, \ldots, t_n)$, and the derivation $T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t$ is of the form

$$\frac{T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t_1 \quad \ldots \quad T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t_n}{T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash \mathsf{f}(t_1, \ldots, t_n)}$$

For all $i \in \{1, \ldots, n\}$, $t_i \in \mathsf{St}(t) \subseteq \mathsf{St}(\mathsf{exec}, sid)$, and by induction hypothesis $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t_i$. We can thus conclude that by application of the corresponding composition rule. We have that:

$$\frac{T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t_1 \quad \ldots \quad T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t_n}{T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash f(t_1, \ldots, t_n)}$$

Case 2 – the last rule is a decomposition rule: We have that the proof tree witnessing $T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t$ is of the form

$$\frac{T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t_1 \quad \ldots \quad T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t_n}{T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t}$$

If $names(t) \subseteq \mathcal{N}_\epsilon$, then we have seen that $T_0 \vdash t$, and thus we conclude. Now, assume that there exists $n_v^{sid'} \in (names(t) \setminus \mathcal{N}_\epsilon) \subseteq names(\mathsf{exec}, sid)$. By Definition of a symbolic trace and of an execution trace (see Definition 5), $n_v^{sid'} \in names(\mathsf{exec}, sid')$. Thus $names(\mathsf{exec}, sid) \cap names(\mathsf{exec}, sid') \neq \emptyset$, and thanks to Lemma 5, we have that: $\mathsf{sameTagAs}(\mathsf{exec}, sid) \;=\; \mathsf{sameTagAs}(\mathsf{exec}, sid')$.

We need to prove that for all $i \in \{1, \ldots, n\}$, $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t_i$. Since π is minimal, we know by locality (Lemma 3) that $t_i \in \mathsf{St}(T_0 \cup \mathcal{N}_\epsilon \cup \mathcal{K}_\epsilon \cup \mathsf{K}(\mathsf{exec})) \cup \mathcal{A} \cup \{\mathsf{pub}(a) \mid a \in \mathcal{A}\}$. We consider two cases:

If $names(t_i) \subseteq \mathcal{N}_\epsilon$, then we have already established that $T_0 \vdash t_i$, and thus $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t_i$.
Otherwise, there exists $n_w^{sid''} \in (names(t_i) \setminus \mathcal{N}_\epsilon)$. In that case, $t_i \in \mathsf{St}(\mathsf{K}(\mathsf{exec}))$, i.e. there exists $k \in \{1, \ldots, \ell\}$ such that $t_i \in \mathsf{St}(\mathsf{e}_k^{sid_k}) \subseteq \mathsf{St}(\mathsf{exec}, sid_k)$; and thus $n_w^{sid''} \in names(\mathsf{exec}, sid_k)$. Moreover, by Definition of a symbolic trace and of an execution trace (see Definition 5), $n_w^{sid''} \in names(\mathsf{exec}, sid'')$. Hence, we have that $names(\mathsf{exec}, sid'') \cap names(\mathsf{exec}, sid_k) \neq \emptyset$, which according to Lemma 5 implies that

$$\mathsf{sameTagAs}(\mathsf{exec}, sid'') \;=\; \mathsf{sameTagAs}(\mathsf{exec}, sid_k).$$

By inspection of the decomposition rules, we note that there must exist $j \in \{1, \ldots, n\}$, such that for all $i \in \{1, \ldots, n\}$, $names(t) \cup names(t_i) \subseteq names(t_j)$, and thus $n_v^{sid'}, n_w^{sid''} \in (names(t_j) \setminus \mathcal{N}_\epsilon)$. Moreover, we have that $t_j \in \mathsf{St}(\mathsf{K}(\mathsf{exec}))$, i.e. there exists $h \in \{1, \ldots, \ell\}$ such that $t_j \in \mathsf{St}(\mathsf{e}_j^{sid_h}) \subseteq \mathsf{St}(\mathsf{exec}, sid_h)$. Hence, $n_v^{sid'}, n_w^{sid''} \in names(\mathsf{exec}, sid_h)$, which according to Lemma 5 implies

$$\begin{array}{rcl} \mathsf{sameTagAs}(\mathsf{exec}, sid') & = & \mathsf{sameTagAs}(\mathsf{exec}, sid_h) \\ \mathsf{sameTagAs}(\mathsf{exec}, sid'') & = & \mathsf{sameTagAs}(\mathsf{exec}, sid_h) \end{array}$$

We therefore can infer that

$$\mathsf{sameTagAs}(\mathsf{exec}, sid) \;=\; \mathsf{sameTagAs}(\mathsf{exec}, sid_k).$$

and by hypothesis on S that $sid_k \in S$. We have thus demonstrated that $t_i \in \mathsf{St}(\mathsf{exec}, sid_k)$ with $sid_k \in S$, which according to our induction hypothesis implies $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t_i$.
Since for all $i \in \{1, \ldots, n\}$, $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t_i$, we can conclude by application of the corresponding decomposition rule that:

$$\frac{T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t_1 \quad \ldots \quad T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t_n}{T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t}$$

□

E.2. **Satisfiability of the formula.**

Lemma 19. Let Π be a k-party protocol, ϕ a closed quantifier-free formula in $\mathcal{L}$, and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a well-formed valid execution of $\widetilde{\Pi}$ that satisfies ϕ, w.r.t. some set T_0 of ground atoms. Moreover, we assume that $T_0 \cup \mathsf{K}(\mathsf{exec}) \not\vdash k$ for any $k \in lgKeys \setminus (\mathcal{K}_\epsilon \cup T_0)$ (exec does not reveal any long term keys). Let S be a set of session identifiers such that:

(1) for all $\mathsf{learn}(t)$ that occurs positively in ϕ such that $t \notin \mathcal{A} \cup lgKeys$, there exists $sid \in S$ such that $t \in \mathsf{St}(\mathsf{exec}, sid)$,
(2) $\mathsf{Ws}(\mathsf{exec}, \phi) \subseteq S$, and
(3) $\forall sess_1, sess_2$ with $\mathsf{ExpectedTag}(\mathsf{exec}, sess_1) = \mathsf{ExpectedTag}(\mathsf{exec}, sess_2)$, we have that
$$sess_1 \in S \text{ if and only if } sess_2 \in S.$$

We have that $\mathsf{exec}|_S$ is an execution of $\widetilde{\Pi}$ that satisfies ϕ, i.e. $\langle \mathsf{exec}|_S, T_0 \rangle \models \phi$.

Proof. We prove this by induction on $(\ell, size(\phi))$ using the lexicographic ordering. Here, ℓ denotes the length (i.e. number of events) of the trace exec and $size(\phi)$ is the size of ϕ (i.e. number of symbols that occur in ϕ without counting the symbol $\neg$ and after elimination of double negation, i.e., $\neg\neg\psi$ is rewritten in ψ).

We need to distinguish several base cases.

Case $|\mathsf{exec}| = 0$**:** In that case $\mathsf{exec}|_S = \mathsf{exec}$, and thus by hypothesis if $\langle \mathsf{exec}, T_0 \rangle \models \phi$, then also $\langle \mathsf{exec}|_S, T_0 \rangle \models \phi$.

Case $\phi = \mathsf{true}$ **(resp.** $\phi = \neg\mathsf{true}$**):** In such a case, we have that $\langle \mathsf{exec}|_S, T_0 \rangle \models \phi$. The case where $\phi = \neg\mathsf{true}$ is impossible.

Case $\phi = \mathsf{Q}(t_1, \ldots, t_n)$**:** If $\langle \mathsf{exec}, T_0 \rangle \models \phi$, then $\mathsf{e}_\ell^{sid_\ell} = \mathsf{Q}(t_1, \ldots, t_n)$, and $\mathsf{Ws}(\mathsf{exec}, \phi) = \{sid_\ell\} \subseteq S$. By Definition 22, $\mathsf{exec}|_S$ ends with the event $\mathsf{Q}(t_1, \ldots, t_n)$. We can thus conclude that $\langle \mathsf{exec}|_S, T_0 \rangle \models \phi$.

Case $\phi = \neg\mathsf{Q}(t_1, \ldots, t_n)$**:** If $\langle \mathsf{exec}, T_0 \rangle \models \neg\mathsf{Q}(t_1, \ldots, t_n)$, then we have that $\mathsf{e}_\ell^{sid_\ell} \neq \mathsf{Q}(t_1, \ldots, t_n)$, and $\mathsf{Ws}(\mathsf{exec}, \phi) = \{sid_\ell\} \subseteq S$ (note that we have already considered the case where $\mathsf{exec} = []$, and thus now we assume that $\mathsf{exec} \neq []$). We have that $\mathsf{exec}|_S$ does not end with $\mathsf{Q}(t_1, \ldots, t_n)$. We can thus conclude that $\langle \mathsf{exec}|_S, T_0 \rangle \models \neg\mathsf{Q}(t_1, \ldots, t_n)$, i.e. $\langle \mathsf{exec}|_S, T_0 \rangle \models \phi$.

Case $\phi = \mathsf{learn}(t)$**:** If $\langle \mathsf{exec}, T_0 \rangle \models \phi$, then $T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash t$. If $t \in \mathcal{A} \cup \mathit{lgKeys}$, since exec doesn't reveal any long-term decryption key, $T_0 \vdash t$, and thus $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t$. If $t \notin \mathcal{A} \cup \mathit{lgKeys}$, then by hypothesis we know there exists $sid \in S$ such that $t \in \mathsf{St}(\mathsf{exec}, sid)$. According to Lemma 6, since by hypothesis $sid \in S$, $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \vdash t$. Hence, we can conclude that $\langle \mathsf{exec}|_S, T_0 \rangle \models \phi$.

Case $\phi = \neg\mathsf{learn}(t)$**:** If $\langle \mathsf{exec}, T_0 \rangle \models \neg\mathsf{learn}(t)$, then $T_0 \cup \mathsf{K}(\mathsf{exec}) \nvdash t$. But since $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \subseteq T_0 \cup \mathsf{K}(\mathsf{exec})$, it is also the case that $T_0 \cup \mathsf{K}(\mathsf{exec}|_S) \nvdash t$, and thus that $\langle \mathsf{exec}|_S, T_0 \rangle \models \neg\mathsf{learn}(t)$.

Case $\phi = \mathsf{C}(u)$**:** If $\langle \mathsf{exec}, T_0 \rangle \models \mathsf{C}(u)$, then we have that $T_0 \vdash \mathsf{priv}(u)$ or $T_0 \vdash \mathsf{shk}(u, v)$ for some $v \neq \epsilon$. Hence, we also have that $\langle \mathsf{exec}|_S, T_0 \rangle \models \mathsf{C}(t)$.

Case $\phi = \neg\mathsf{C}(u)$**:** If $\langle \mathsf{exec}, T_0 \rangle \models \neg\mathsf{C}(u)$, then we have that $T_0 \nvdash \mathsf{priv}(u)$ and $T_0 \nvdash \mathsf{shk}(u, v)$ for all $v \neq \epsilon$. Hence, we also have that $\langle \mathsf{exec}|_S, T_0 \rangle \models \neg\mathsf{C}(t)$.

We distinguish several inductive cases ($|\mathsf{exec}| > 1$ and $\mathit{size}(\phi) > 1$).

Case $\phi = \phi_1 \vee \phi_2$**:** If $\langle \mathsf{exec}, T_0 \rangle \models \phi$ then $\langle \mathsf{exec}, T_0 \rangle \models \phi_1$ or else $\langle \mathsf{exec}, T_0 \rangle \models \phi_2$. Assume that $\langle \mathsf{exec}, T_0 \rangle \models \phi_1$ (the other case can be done in a similar way). It is easy to see that the three conditions needed to apply our inductive hypothesis are fulfilled. We can thus apply our inductive hypothesis on ϕ_1 to conclude that $\langle \mathsf{exec}|_S, T_0 \rangle \models \phi_1$, and thus that $\langle \mathsf{exec}|_S, T_0 \rangle \models \phi_1 \vee \phi_2$.

Case $\phi = \neg(\phi_1 \vee \phi_2)$**:** If $\langle \mathsf{exec}, T_0 \rangle \models \neg(\phi_1 \vee \phi_2)$, then $\langle \mathsf{exec}, T_0 \rangle \models \neg\phi_1$ and $\langle \mathsf{exec}, T_0 \rangle \models \neg\phi_2$. Again, the three conditions needed to apply our inductive hypothesis are fulfilled. We can thus apply our inductive hypothesis to conclude that $\langle \mathsf{exec}|_S, T_0 \rangle \models \neg\phi_1$ and $\langle \mathsf{exec}|_S, T_0 \rangle \models \neg\phi_2$, and thus $\langle \mathsf{exec}|_S, T_0 \rangle \models \neg(\phi_1 \vee \phi_2)$.

Case $\phi = \Diamond\psi$**:** If $\langle \mathsf{exec}, T_0 \rangle \models \phi$, we know that there exists $i \in \{1, \ldots, \ell\}$ such that $\langle \mathsf{exec}_i, T_0 \rangle \models \psi$ and $\mathsf{Ws}(\mathsf{exec}, \phi) = \mathsf{Ws}(\mathsf{exec}_i, \psi)$.

- Let $\mathsf{learn}(t)$ be a subformula that occurs positively in ψ such that $t \notin \mathcal{A} \cup \mathit{lgKeys}$. Then, by definition, $\mathsf{learn}(t)$ also occurs positively in ϕ, and thus by hypothesis, there exists $sid \in S$ such that $t \in \mathsf{St}(\mathsf{exec}, sid)$.
- We have that $\mathsf{Ws}(\mathsf{exec}, \phi) = \mathsf{Ws}(\mathsf{exec}_i, \psi)$, and by hypothesis $\mathsf{Ws}(\mathsf{exec}, \phi) \subseteq S$. Thus $\mathsf{Ws}(\mathsf{exec}_i, \psi) \subseteq S$.
- By hypothesis, we have that S is such that for all sess_1 and sess_2 such that $\mathsf{ExpectedTag}(\mathsf{exec}, \mathit{sess}_1) = \mathsf{ExpectedTag}(\mathsf{exec}, \mathit{sess}_2)$, $\mathit{sess}_1 \in S$ if and only if $\mathit{sess}_2 \in S$.

The three conditions are fulfilled, we can thus apply our inductive hypothesis to conclude that $\mathsf{exec}_i|_S$ also satisfies ψ, i.e. $\langle \mathsf{exec}_i|_S, T_0\rangle \models \psi$. But then there exists j such that $\mathsf{exec}_i|_S = (\mathsf{exec}|_S)_j$, and thus such that $\langle (\mathsf{exec}|_S)_j, T_0\rangle \models \psi$, which according to the semantics of $\mathcal{L}$ gives us $\mathsf{exec}|_S$ satisfies $\Diamond\psi$, i.e. $\langle \mathsf{exec}|_S, T_0\rangle \models \Diamond\psi$.

Case $\phi = \neg\Diamond\psi$: If $\langle \mathsf{exec}, T_0\rangle \models \neg\Diamond\psi$, then according to the semantics of $\mathcal{L}$, we have that $\langle \mathsf{exec}_{\ell-1}, T_0\rangle \models \neg\Diamond\psi$ and $\langle \mathsf{exec}, T_0\rangle \models \neg\psi$.

- In the syntax of $\mathcal{L}$, see Definition 8, $\mathsf{learn}(t)$ must not occur under a modality, so the first condition is trivially fulfilled.
- By definition, $\mathsf{Ws}(\mathsf{exec}, \phi) = \emptyset \subseteq S$.
- By hypothesis, we have that S is such that for all $sess_1$ and $sess_2$ such that $\mathsf{ExpectedTag}(\mathsf{exec}, sess_1) = \mathsf{ExpectedTag}(\mathsf{exec}, sess_2)$, $sess_1 \in S$ if and only if $sess_2 \in S$.

We apply our inductive hypothesis and conclude that $\langle (\mathsf{exec}_{\ell-1})|_S, T_0\rangle \models \neg\Diamond\psi$. Now, we distinguish two cases: either $sid_\ell \in S$ or $sid_\ell \notin S$. In the first case, we can also apply our inductive hypothesis on $\langle \mathsf{exec}, T_0\rangle \models \neg\psi$ (note that $\mathsf{Ws}(\mathsf{exec}, \psi) \subseteq \{sid_\ell\} \subseteq S$ since ψ is from the restricted syntax according to Definition 8) and conclude that $\langle \mathsf{exec}|_S, T_0\rangle \models \neg\psi$. This allows us to conclude that $\langle \mathsf{exec}|_S, T_0\rangle \models \neg\Diamond\psi$. In the second case, we have that $\mathsf{exec}|_S = \mathsf{exec}_{\ell-1}|_S$, and thus we conclude that $\langle \mathsf{exec}|_S, T_0\rangle \models \neg\Diamond\psi$. □

Lemma 20. Let Π be a k-party protocol, $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a valid execution of $\widetilde{\Pi}$ w.r.t. some set T_0 of ground atoms, $\phi = \exists x_1. \ldots . \exists x_n. \psi$ be an attack formula of $\mathcal{L}$ (see Definition 10), $\sigma = \{x_1 \mapsto m_1, \ldots, x_n \mapsto m_n\}$ be a ground substitution, S be a set of session identifiers such that $\mathsf{Ws}(\mathsf{exec}, \psi\sigma) \subseteq S$, and $n_\epsilon^\epsilon \in \mathcal{N}_\epsilon$ be an intruder nonce not appearing in exec. If $\langle \mathsf{exec}, T_0\rangle \models \psi\sigma$ then we have that $\langle \mathsf{exec}, T_0\rangle \models \psi\sigma'$ where for all $j \in \{1, \ldots, n\}$

$$\sigma'(x_j) = \begin{cases} n_\epsilon^\epsilon & \text{if } \sigma(x_j) \notin \mathsf{St}(\mathsf{exec}, S) \cup \mathcal{A} \cup \mathit{lgKeys} \cup \mathcal{N}_\epsilon \cup \mathcal{K}_\epsilon \\ \sigma(x_j) & \text{otherwise} \end{cases}$$

Proof. We prove this result by induction on $(\ell, \mathit{size}(\psi))$ using the lexicographic ordering where ℓ denotes the length of the trace exec, and $\mathit{size}(\psi)$ the size of ψ (i.e. number of symbols that occur in ψ without counting the symbol $\neg$ and after elimination of double negation, i.e., $\neg\neg\psi$ is rewritten in ψ). Actually, we strengthen the induction hypothesis by only requiring the hypothesis $\mathsf{Ws}(\mathsf{exec}_p, \psi\sigma) \subseteq S$ when some status event occurs positively in $\psi\sigma$.

Base case ($\mathit{size}(\psi) = 1$) We distinguish several base cases.

- *Case* $\psi' = \mathsf{true}$. In that case $\psi\sigma = \psi\sigma' = \mathsf{true}$ and we easily conclude.
- *Case* $\psi' = \neg\mathsf{true}$. This case is impossible since such a formula is not satisfiable.
- *Case* $\psi = \mathsf{Q}(t_1, \ldots, t_h)$. In that case, we have that $\psi\sigma = \mathsf{Q}(t_1\sigma, \ldots, t_h\sigma)$, and $\mathsf{e}_p^{sid_p} = \mathsf{Q}(t_1\sigma, \ldots, t_h\sigma)$ with $sid_p \in S$. But then, by Definition of σ', we have that $\mathsf{e}_p^{sid_p} = \mathsf{Q}(t_1\sigma, \ldots, t_h\sigma) = \mathsf{Q}(t_1\sigma', \ldots, t_h\sigma')$, and we conclude that $\langle \mathsf{exec}_p, T_o\rangle \models \psi\sigma'$.
- *Case* $\psi = \neg\mathsf{Q}(t_1, \ldots, t_h)$. In that case, $\psi\sigma = \mathsf{Q}(t_1\sigma, \ldots, t_h\sigma)$, and either $\mathsf{exec}_p = []$ or $\mathsf{e}_p^{sid_p} \neq \mathsf{Q}(t_1\sigma, \ldots, t_h\sigma)$. In the first case, according to the semantics of our logic $\mathcal{L}$, we conclude that $\langle \mathsf{exec}_p, T_0\rangle \models \psi\sigma'$ $(= \neg\mathsf{Q}(t_1\sigma', \ldots, t_h\sigma'))$. In the second case, i.e. $\mathsf{e}_p^{sid_p} \neq \mathsf{Q}(t_1\sigma, \ldots, t_h\sigma)$, by Definition of σ', we have that $t_k\sigma' \in \{t_k\sigma, n_\epsilon^\epsilon\}$

for any $k \in \{1, \ldots, h\}$. Hence, we have that $\mathsf{e}_p^{sid_p} \neq \mathsf{Q}(t_1\sigma', \ldots, t_h\sigma')$, and thus $\langle \mathsf{exec}_p, T_0 \rangle \models \psi\sigma'$ $(= \neg\mathsf{Q}(t_1\sigma', \ldots, t_h\sigma'))$.

- *Case* $\psi = \mathsf{learn}(t)$. In that case $t\sigma' \in \{t\sigma, n_\epsilon^\epsilon\}$ (thanks to Condition 1 of Definition 10), then we know by hypothesis that $\mathsf{K}(\mathsf{exec}_p) \cup T_0 \vdash t\sigma'$ and thus, we conclude.
- *Case* $\psi = \neg\mathsf{learn}(t)$. This case cannot occur because ψ satisfies the conditions of an attack formula (see Definition 10), and in particular no $\mathsf{learn}(u)$ appears negatively in ψ.
- *Case* $\psi = \mathsf{C}(t)$ *or* $\neg\mathsf{C}(t)$. In that case, we have that $t\sigma \in \mathcal{A}$. By construction, we have that $t\sigma = t\sigma'$, and this allows us to conclude.

We distinguish several inductive cases.

- *Case* $\psi = \psi_1 \vee \psi_2$. Assume that $\langle \mathsf{exec}_p, T_0 \rangle \models \psi_1\sigma$. The case where $\langle \mathsf{exec}_p, T_0 \rangle \not\models \psi_1\sigma$ but $\langle \mathsf{exec}_p, T_0 \rangle \models \psi_2\sigma$ can be proved in a similar way. By definition, we have that $\mathsf{Ws}(\mathsf{exec}_p, \psi_1\sigma) = \mathsf{Ws}(\mathsf{exec}_p, \psi\sigma) \subseteq S$. We can thus apply our inductive hypothesis to conclude that $\langle \mathsf{exec}_p, T_0 \rangle \models \psi_1\sigma'$ and thus $\langle \mathsf{exec}_p, T_0 \rangle \models \psi\sigma'$ $(= \psi_1\sigma' \vee \psi_2\sigma')$.
- *Case* $\psi = \neg(\psi_1 \vee \psi_2)$. In that case, $\langle \mathsf{exec}_p, T_0 \rangle \models \neg\psi_1\sigma$ and $\langle \mathsf{exec}_p, T_0 \rangle \models \neg\psi_2\sigma$. By definition, we have that:

  $$\mathsf{Ws}(\mathsf{exec}_p, \neg\psi_1\sigma) \cup \mathsf{Ws}(\mathsf{exec}_p, \neg\psi_2\sigma) = \mathsf{Ws}(\mathsf{exec}_p, \psi\sigma) \subseteq S.$$

  By applying our inductive hypothesis, we obtain that $\langle \mathsf{exec}_p, T_0 \rangle \models \neg\psi_j\sigma'$ for $j \in \{1, 2\}$. This allows us to conclude that $\langle \mathsf{exec}_p, T_0 \rangle \models \psi\sigma'$ $(= \neg(\psi_1\sigma' \vee \psi_2\sigma'))$.
- *Case* $\psi = \Diamond\psi'$. In that case, according to the semantics of our logic $\mathcal{L}$, there exists $j \leq i$ such that $\langle \mathsf{exec}_j, T_0 \rangle \models \psi'\sigma$, and thus by inductive hypothesis we know that $\langle \mathsf{exec}_j, T_0 \rangle \models \psi'\sigma'$. Hence, we have that $\langle \mathsf{exec}_p, T_0 \rangle \models \psi\sigma'$ $(= \Diamond\psi'\sigma')$.
- *Case* $\psi = \neg\Diamond\psi'$. In that case, according to the semantics of our logic, we have that $\langle \mathsf{exec}_{p-1}, T_0 \rangle \models \psi\sigma$ and $\langle \mathsf{exec}_p, T_0 \rangle \models \neg\psi'\sigma$. By inductive hypothesis we know that $\langle \mathsf{exec}_{p-1}, T_0 \rangle \models \psi\sigma'$. Note that, by definition of an attack formula (see Definition 10), there is no positive status event in $\psi\sigma$. Moreover, using our inductive hypothesis, we obtain that $\langle \mathsf{exec}_p, T_0 \rangle \models \neg\psi'\sigma'$ (note that, again, by definition of an attack formula, we know that there is no positive status event in $\neg\psi'\sigma$). This allows us to conclude that $\langle \mathsf{exec}_p, T_0 \rangle \models \psi\sigma'$ $= (\neg\Diamond\psi'\sigma')$. □

Lemma 7. Let Π be a k-party protocol, and $\mathsf{exec} = [\mathsf{e}_1^{sid_1}; \ldots; \mathsf{e}_\ell^{sid_\ell}]$ be a valid and well-formed execution of $\widetilde{\Pi}$ w.r.t. some set T_0 of ground atoms such that $T_0 \cup \mathsf{K}(\mathsf{exec}) \not\vdash k$ for any $k \in \mathit{lgKeys} \setminus (\mathcal{K}_\epsilon \cup T_0)$. Let $\phi = \exists x_1. \ldots . \exists x_n. \psi$ be an attack formula of $\mathcal{L}$, and σ be a ground substitution such that $\langle \mathsf{exec}, T_0 \rangle \models \psi\sigma$. Let S be a set of session identifiers such that:

(1) $\mathsf{Ws}(\mathsf{exec}, \psi\sigma) \subseteq S$, and
(2) $\forall \mathit{sess}_1, \mathit{sess}_2$ with $\mathsf{ExpectedTag}(\mathsf{exec}, \mathit{sess}_1) = \mathsf{ExpectedTag}(\mathsf{exec}, \mathit{sess}_2)$, we have that
$$\mathit{sess}_1 \in S \text{ if and only if } \mathit{sess}_2 \in S.$$

We have that $\mathsf{exec}|_S$ is an execution of $\widetilde{\Pi}$ that satisfies ϕ, i.e. $\langle \mathsf{exec}|_S, T_0 \rangle \models \phi$.

Proof. First, we apply Lemma 20 to ensure that the substitution σ witnessing the fact that the attack formula ϕ is satisfiable only uses atomic terms and subterms that occur in $\mathsf{St}(\mathsf{exec}, S)$. Hence, thanks to this lemma, we can assume w.l.o.g. that for all $j \in \{1, \ldots, n\}$, $\sigma(x_j) \in \mathsf{St}(\mathsf{exec}, S) \cup \mathcal{A} \cup \mathit{lgKeys} \cup \mathcal{N}_\epsilon \cup \mathcal{K}_\epsilon$. Then, we apply Lemma 19 in order to conclude. □